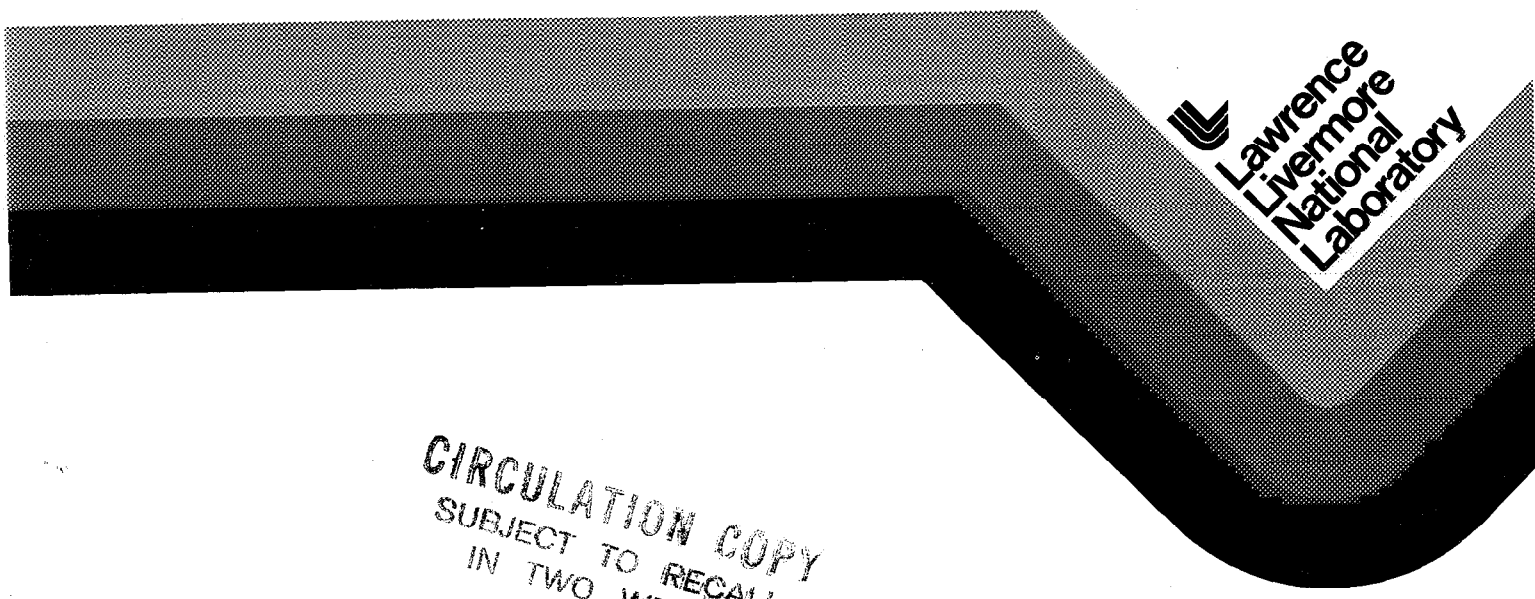
UCRL-52995

Verification of timing constraints on large digital systems

Thomas Melvin McWilliams

(Ph.D. Thesis)

May 1980



Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under Contract W-7405-Eng-48.

UCRL-52995

Verification of timing constraints on large digital systems

Thomas Melvin McWilliams

(Ph.D. Thesis)

Manuscript date: May 1980

LAWRENCE LIVERMORE LABORATORY
University of California • Livermore, California • 94550

Available from: National Technical Information Service • U.S. Department of Commerce
5285 Port Royal Road • Springfield, VA 22161 • $9.00 per copy • (Microfiche $3.50)

VERIFICATION OF TIMING CONSTRAINTS ON LARGE DIGITAL SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

by

Thomas Melvin McWilliams

May 1980

Verification of Timing Constraints in Large Digital Systems

Thomas Melvin McWilliams

May 1980

Computer Science Department
Stanford University
Stanford, California 94305

ABSTRACT

A new approach to the verification of timing constraints on large digital systems has been developed. The associated algorithm is computationally efficient, and provides early and continuous feedback about the timing aspects of synchronous sequential circuits as they are designed. It also provides means for conveniently verifying the design section-by-section for designs which are too large to examine as a unit.

This approach is new in that it uses a "stable value" to represent signals in the large majority of instances in which it is unnecessary to know whether the signals are true or false in order to examine satisfaction of the timing constraints. For the remaining instances, it represents the full value behavior of signals, allowing it to evaluate compliance with the remaining timing constraints. This use of the "stable value" greatly reduces the number of states through which a digital system needs to be taken in the process of verifying its timing constraints, which in turn greatly reduces the amount of computing effort required, relative to that needed to verify the timing constraints via more traditional logic simulation. Not needing to know the values of most signals also greatly reduces the size of the data base needed to drive the verification process, relative to that required in doing logic simulation. Both of these savings are of exponential order. This approach thus makes feasible for the first time the exhaustive examination of complex digital circuits for satisfaction of timing constraints.

A system has been implemented using this approach which takes a digital logic design specified in the SCALD Hardware Description Language, and verifies all of the timing constraints specified within it. This system has been used in the design of a very high performance central processing unit, the S-1 Mark IIA processor. The use of the Timing Verifier allowed timing errors to be identified early in the design process, while it was still easy to correct them. Such timely error elimination has permitted the design to be completed more rapidly, and has also supported the creation of a design which will perform more rapidly without timing errors, when it is implemented in hardware.

KEYWORDS: Design verification, Timing constraints, Digital systems, Logic simulation, Hierarchical design

Acknowledgments

I would like to thank Forest Baskett, Bill vanCleemput, and Lowell Wood for the constant support and helpful guidance they have provided throughout the course of this research, and for their many useful suggestions for improvements in this thesis. The Fannie and John Hertz Foundation's gracious support via a Hertz Fellowship through the course of my graduate studies has provided me the freedom to pursue this research. Curt Widdoes worked with me on the development of the SCALD base on which the Timing Verifier has been erected, in a collaboration that was as productive as it was enjoyable. Bill Bryson, Mike Farmwald, and Jeff Rubin have been the first major users of the Timing Verifier, and their patience and many suggestions for its improvement have been most appreciated. Steve Correll generously provided invaluable aid in building and enhancing the S-1 documentation system, on which this thesis was produced.

My thanks also go to the Office of Naval Research, the Naval Electronics System Command, and the Naval Material Command for their support of the S-1 Project, which has provided the necessary environment for this research, and to Curt Widdoes and Lowell Wood, whose tireless efforts were essential in getting the S-1 Project underway.

Last, but most fervently of all, my gratitude to my wife, Beth, for having put up with me during this period.

This work was in part performed under the auspices of the U.S. Department of Energy at the Lawrence Livermore National Laboratory, which is operated by the University of California under contract No. W-7405-ENG-48.

Table of Contents

List of Figures

List of Tables

Chapter I

INTRODUCTION

1.1 PURPOSE OF THIS INVESTIGATION

The components comprising a digital system have a range of switching times associated with them. If the voltage at one end of a wire is changed, then the voltage at the other end of the wire will change after some delay, which is partly determined by the transmission line characteristics of the wire and its length. If the input of a gate or other circuit is changed, there is a delay time which must elapse before its output can be guaranteed to have the value corresponding to its new input value. Because of variations in the construction of these components, these delay times vary from one component to the next.

In order that a digital system perform correctly, a designer must take into account the possible propagation delays associated with each of the elements making up the system. If a path through a digital system has either too long or too short a delay associated with it, then the value of the circuit may be wrong at a critical point in space and time, causing the circuit to yield an incorrect result. This is called a timing error. Digital logic as it is currently implemented is intrinsically susceptible to such errors, and their complete elimination from all portions of a digital logic system is essential to guarantee that the logic will perform reliably and reproducibly under all variations in data and programs. This thesis addresses the early and efficient detection of these timing errors, so that digital logic designers can henceforth frequently check their designs for these errors as the design proceeds, thereby finding timing errors before the design proceeds too far and the errors become difficult to correct. Indeed, for complicated logic circuits, the ideas developed in this thesis makes exhaustive timing verification feasible for the first time.

A system which uses these ideas has been implemented and is called the SCALD Timing Verifier. The Timing Verifier is a part of the SCALD (Structured Computer-Aided Logic Design) system [Mc78a, Mc78b, Sp78, Sp79]. SCALD is a complete computer-aided design automation environment which processes a graphics-based, hierarchical description of a digital logic design, generating a complete set of low-level documentation which includes that necessary to implement it in hardware.

The timing verification approach developed here operates on synchronous

sequential systems. It performs a complete timing constraint verification based on the minimum and maximum propagation delays of the circuit components, their set-up and hold times, minimum pulse width constraints, and interconnection delays.

One of the principal features of this approach is its ability to verify designs by modules, where a module is a logical section of a design. All of the signals going between modules must have user-specified assertions on them stating when they can change, and when they are stable. This ability to verify designs by modules permits much larger designs to be verified than would otherwise be possible because of limitations on the amount of memory available. It is also convenient for the verification of designs being done by a group of designers, to allow each designer to verify the timing constraints within his section of the design, independently of the rest of the design.

The utilization of user-specified timing assertions on not-yet-generated signals allows the design to be checked as it progresses, even on a day-by-day basis. This is particularly important in that it allows timing errors to be corrected before they have a chance to propagate their effects throughout the design, or to cause major changes to be made late in the design. It also supports formation of an accurate estimate of the cycle time of a digital system before its design is completed.

1.2 TYPES OF DIGITAL SYSTEMS

Digital logic systems can be classified into the following different types: combinational systems, synchronous sequential systems, or asynchronous sequential systems. The following sections will define these different types of systems, and will discuss the types of timing errors that occur in them.

1.2.1 Combinational Systems

Digital systems whose outputs are only a function of the current values of the inputs are called combinational systems [Kr67, Mc62, Ob70, Un69]. These systems have no internal state, and tend to be fairly simple. They consist of some number of levels of gates and inverters connected together with no feedback paths.

The main timing parameters of interest for combinational systems are the minimum and maximum delay through the logic from each of the inputs to each of the outputs. These are the timing parameters normally given by the manufacturer for an integrated circuit which consists of a combinational network of gates.

The calculation of the minimum propagation delay from any one input to any output is fairly simple. The delay along each of the paths from the input to the output of interest is calculated by summing up the minimum delay of each of the elements along the path, and then the minimum delay is given by the shortest delay path. The maximum propagation delay is calculated in the same way, except that the maximum delay of each element is used, and then the longest delay path determines the maximum propagation delay.

1.2.2 Synchronous Sequential Systems

A digital system is *sequential* if it stores information concerning its past input states. A *synchronous sequential* system [Kr67, Mc62, Ob70, Un69] is one in which the stored internal state changes only at times determined by a central clock. The internal state is either stored in registers or latches, but is never stored by just creating feedback paths within the logic. In fact, every feedback path must contain one or more clocked registers or latches. Figure 1-1 shows a block diagram of a synchronous sequential system.

A synchronous sequential system must have one central clock. All of the clock signals used in the design must then be generated from this central clock. The reason for having only one clock is so that the entire network uses a common standard for

determining when to change the stored state. If multiple clocks are used, then a section of the logic which uses one clock cannot be certain that signals generated from a section which uses another clock will be stable at a particular time within its clock period, even if all relevant timing properties of the two sections are known. This problem of communicating between two synchronous systems is the classical synchronization problem [Li66, Co69, Ch72, Hu75].

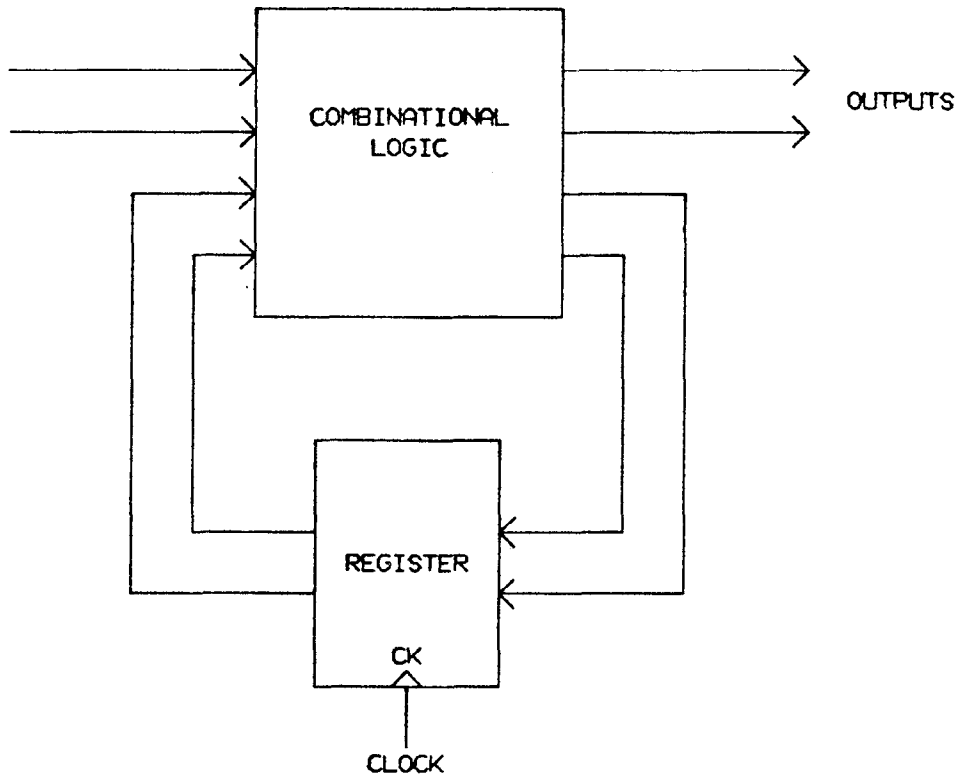


Figure 1-1

A synchronous sequential system

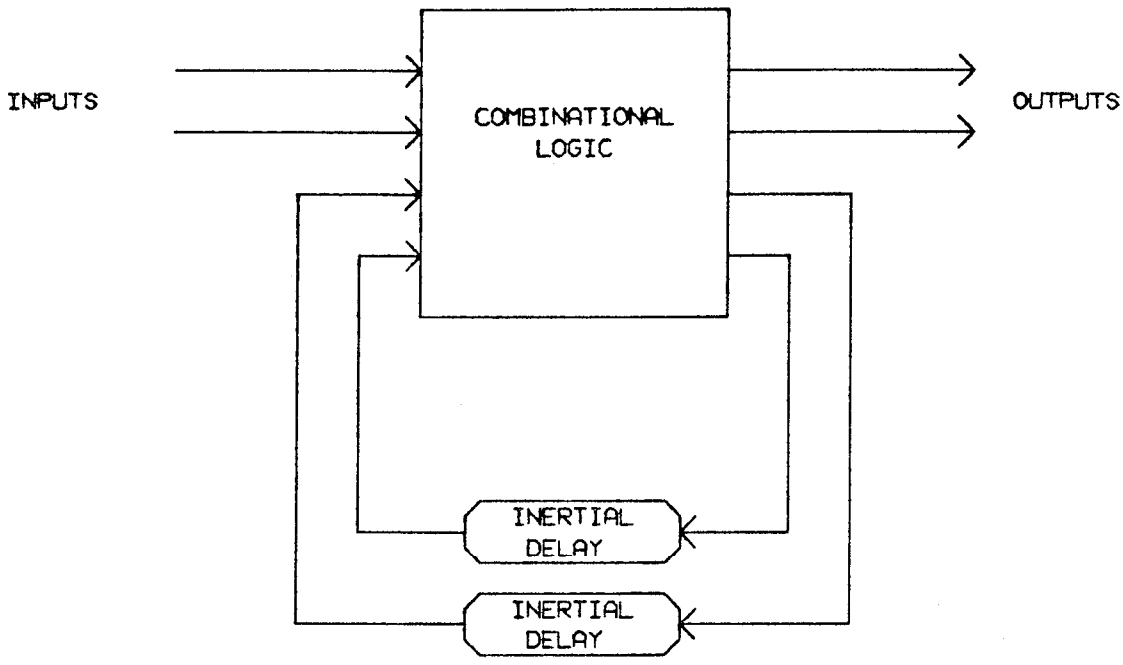


Figure 1-2

An asynchronous sequential system

The following example demonstrates the operation of a simple asynchronous sequential circuit. Figure 1-3 shows a set-reset type latch constructed out of two NOR gates. Normally both the "SET" and "RESET" inputs are false. If the "SET" input goes true, then the "A" output will go false, which in turn will cause the "B" output to go true. With the "B" output true, the "A" output will stay false even if the "SET" input is made

1.2.3 Asynchronous Sequential Systems

Figure 1-2 shows a block diagram of an asynchronous sequential system. Asynchronous sequential systems store internal state, like synchronous sequential systems, but are not required to have a central clock to control when the stored internal state can change [Kr67, Mc62, Ob70, Un69]. Both synchronous sequential systems and combinational systems are special cases of asynchronous sequential systems.

Asynchronous sequential systems can either use clocked registers and latches to hold the internal state, or they can store state information by having feedback paths within the logic network which contain some delay in them. The delay is necessary so that when an input changes, the *new* outputs are a function of the *old* output values (and the *new* inputs), and not of the *new* outputs. The inertial-delay elements shown in the feedback paths in Figure 1-2 also filter out small pulses which may occur in the output from the combinational logic elements. This filtering is necessary if the combinational logic contains any *hazards*. (A *hazard* occurs when a signal goes to the wrong value for a short period of time, because of a difference in delay between different paths through the combinational logic.)

false again. In this way, the latch has stored the information that the "SET" input has been true. The "RESET" input can be used to clear the output of the latch in a similar fashion. The delay in the feedback path is provided by the intrinsic internal delay of the gates which constitute the circuit.

Determining when one signal is changing in relation to another signal in asynchronous sequential circuits is much more difficult than in synchronous sequential circuits, because there is no central clock determining when signals can be changed. Instead, there are numerous delays and feedback paths within the network controlling the timing of the circuit. Verifying that there are no timing errors in an asynchronous sequential circuit is therefore fundamentally more difficult than doing so for synchronous sequential circuits.

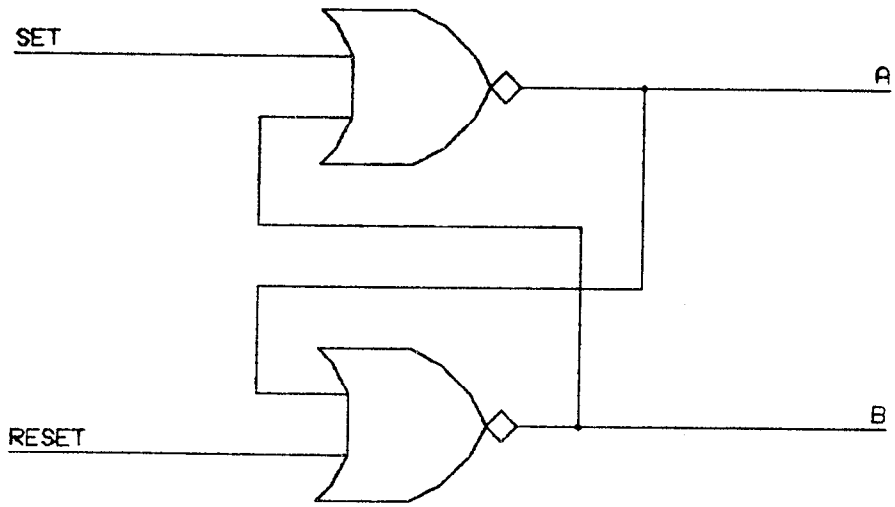


Figure 1-3

A set-reset latch

1.2.4 Types of Systems Addressed by this Thesis

The timing verification approach developed in this thesis is designed to operate on combinational and synchronous sequential circuits. These are the dominant types of circuits used in large digital systems at the present time. In systems which contain a mixture of circuit types, this approach may be used to verify correct timing behavior of the synchronous and combinational parts, ignoring the rest of the design. Analysis of the timing of asynchronous circuits requires full functional verification, which is beyound the scope of this thesis.

1.3 TYPES OF TIMING ERRORS MADE BY DESIGNERS

Within synchronous sequential digital systems, there are a number of different levels at which timing errors may occur. These can be resolved into three main levels: system-level timing errors which occur over multiple clock cycles, logic-level timing errors which occur within a clock cycle, and circuit-level timing errors which occur within a gate or storage element. The next three sections will treat each type of error in detail.

1.3.1 System-Level Timing Errors

On the system level, timing errors may exist in both the software running on the system, in the microcode and in the hardware. System-level timing errors are those types of errors which occur between two units interacting over multiple clock cycles.

Consider the timing errors which can occur in software. The operating system might try and read some data being retrieved from a disk storage unit before the unit's controller has finished writing it into the CPU's memory. This results in the wrong value being read. Another example is an interrupt occurring during a critical sequence of code. This could result in an input operation changing a variable which the code sequence depended on not being changed at that time.

In the microcode or hardware, there can be errors in the communications protocol used between two different units. Consider a CPU talking to a controller on a bus. The definition of the bus might require that the CPU wait until an acknowledgment signal is generated before it is allowed to proceed after a particular operation. If there is a design error in the CPU such that it waits a certain amount of time, instead of waiting for the acknowledgment signal, an error would occur if the acknowledgment signal arrived later than expected.

Verification of these system-level timing errors is beyond the scope of this thesis. The currently best known way of addressing these problems is through the use of logic or system level simulation.

1.3.2 Logic-Level Timing Errors

On the logic level, there are a number of types of timing errors which may occur. These include failure to meet the set-up, hold, or minimum pulse width time requirements for a register, latch, memory, or other complex function. These generally occur because the delay of the combinational logic between two clocked elements is too long or too short. In addition, problems due to hazards on clock signals may arise, resulting in a register or latch being clocked unexpectedly. This could possibility cause data to be lost. The actual delays of the interconnections between the components are also a significant consideration, accounting for as much as half the delay in current large systems.

Figure 1-4 shows an edge-triggered D-type register, along with the definition of the propagation delay, set-up time, hold time, and minimum pulse width constraints. When the clock input of this register goes from a zero to a one, the register will change its output from its current value to the value given by its data input, after its propagation delay has elapsed. If the data input is changing during the rising edge of

the clock, then the value to which the register will be set is indeterminate. In order to insure that the register is set to the proper value, the data input must be stable for a period before the rising edge of the clock (*the set-up time*), and it must remain stable for a period after the rising edge of the clock (*the hold time.*) In addition to set-up and hold time constraints, the register may not operate properly unless the clock pulse is at least as wide as the minimum clock pulse width specified for it.

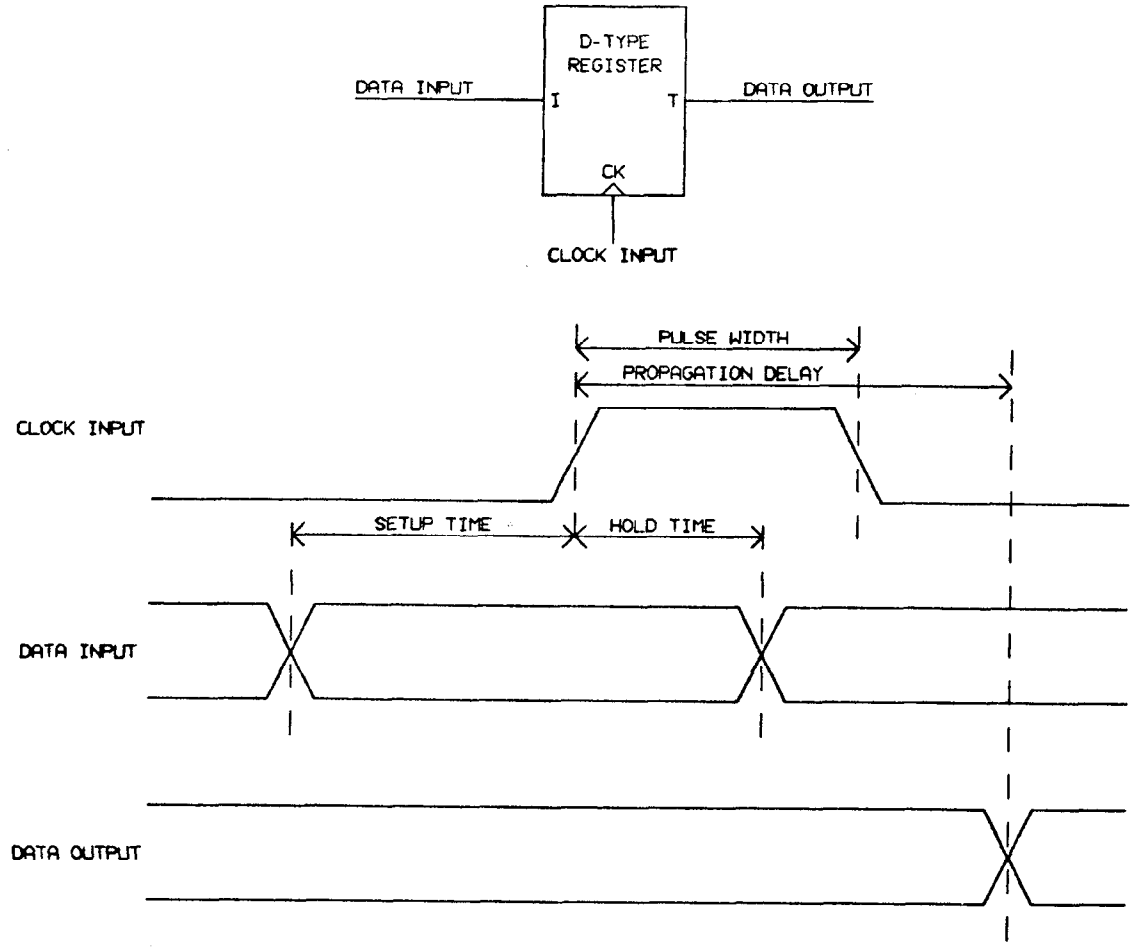


Figure 1-4

Example of a D-Type register

Figure 1-5 shows an edge-triggered register being clocked by the output of a gate which has a hazard on it. The intent of this circuit is to conditionally clock the register

based on the value of the signal "ENABLE", but because of too much delay in the generation of "ENABLE", the register occasionally gets clocked when it is not supposed to. In Figure 1-5, the signal "CLOCK" goes from a zero to a one 20 nsec into the cycle, and back to a zero 30 nsec into the cycle. The signal "ENABLE" wants to be a zero, in order to inhibit the register from being clocked, but doesn't get to a zero until 25 nsec into the cycle. As a result, the signal "REG CLOCK" is a short, 5 nsec pulse, which may clock the register, rather than staying zero. This example is typical of a whole class of common timing errors where control signals are generated too late to reliably control the clocking of a register, latch, or memory element. The result can be a circuit that usually works, but will occasionally fail, e.g., when a clock pulse wide enough to cause the register to clock gets through when it is not supposed to. This type of intermittent timing error can be particularly hard to find after the system is constructed, and can result in systems that operate unreliably, but are nearly incapable of being fixed.

A significant consideration in the design of large digital systems arises from the various delays in the interconnections between the logic elements. For short interconnections, a timing performance analysis needs to look at the length, capacitance and inductance of each interconnection in order to determine both the minimum and maximum possible delay. For interconnections having propagation times longer than roughly a quarter period of the voltage wave, a detailed analysis of the transmission line characteristics is required to determine the minimum and maximum possible delay, and whether there are any voltage wave reflections from impedance variations in the signal run of sufficient magnitude to cause extra clock transitions to occur, possibly causing a register to get clocked more times than is intended. Runs with such reflections on them

can be flagged by the transmission line simulator, allowing the timing verification process to flag them if they affect edge-sensitive inputs.

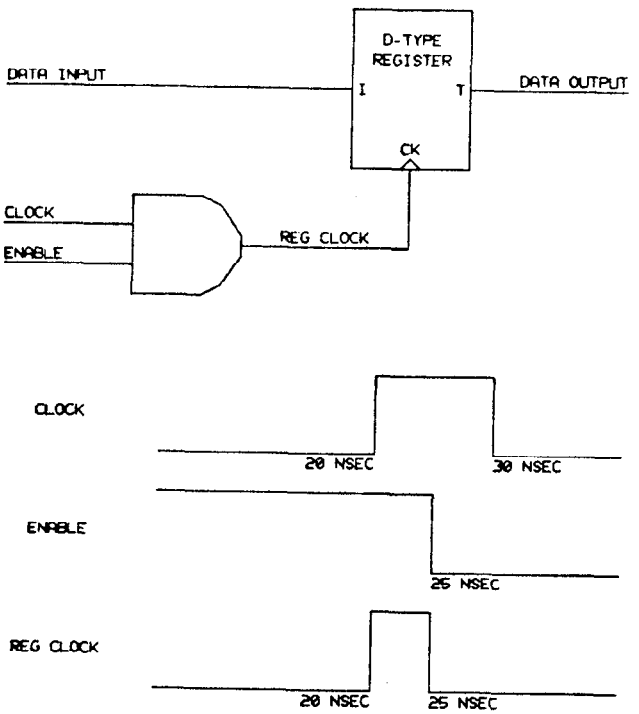


Figure 1-5

Example of a hazard on a clock input to a register

1.3.3 Circuit-Level Timing Errors

The circuit-level timing errors are those occurring within the design of the low-level circuits which implement the basic gate and storage elements.

The analysis of the timing properties of the basic circuits used to implement the gates and storage elements requires consideration of the analog characteristics of the transistors and other devices used to construct them. Timing is determined by the current-driving capability of the transistors, their frequency characteristics and the amount of line and load capacitance that they are required to drive. Timing analysis must determine the minimum and maximum propagation delays from the inputs to the outputs. For registers, latches, and memory elements, values must be determined for the set-up, hold, and minimum pulse width constraints which will insure reliable operation of the circuit.

The analysis of these low-level circuits requires detailed circuit analysis, which is beyound the scope of this thesis. The technique developed in this thesis verifies a design in terms of parameterized models that represent the timing properties of these low-level circuits. The parameters for these models are normally specified by integrated circuit manufacturers when designing with standard components, or can be determined

through low-level circuit analysis for custom integrated circuits.

1.4 PREVIOUS APPROACHES TO TIMING VERIFICATION

There have been a number of approaches to the verification of timing constraints in digital logic systems. They can be grouped into two main categories: *logic simulation* and *worst-case path analysis*. The next two sections will review these two approaches.

1.4.1 Logic Simulation

The logic simulation approach consists of building a model of a digital system, which represents both its logical and timing properties, and then using this model to detect both logic and timing errors. This approach is currently widely used [Ba78, Bo77, Br72a, Ch74, Ch75a, Ch75b, Ha69, Kr77, Ku76, Ma77a, Ma77b, Sz72, Sz75]. If the system being simulated is a computer, then often programs will be loaded into the simulator to be executed to determine if the digital logic design being examined will execute them correctly and, if not, why not. The advantage of this approach over directly implementing a prototype and testing it is that it is generally easier, faster and cheaper to identify and correct the errors in the simulated design, and to then update the

simulation model to reflect these changes. The logic simulator can also be built to take into account variations in the propagation delays, set-up times, hold-times, and minimum pulse width constraints, all of which occur from one component to the next. A hardware prototype, on the other hand, only represents one sample of the large number of possible combinations of timing properties, and cannot test the effects of possible variations in the components' timing properties.

The logic simulation of a circuit design can only show that the cases simulated will work properly when the design is implemented. Therefore, unless all possible cases which have distinct timing paths for a design can be simulated, there is no guarantee that it does not contain undetected timing errors. For the design of a computer system, this requires that all possible programs that exercise distinct timing paths need to be identified and tried, if it is to be definitively shown that there are no timing errors in the design. This is clearly a difficult task for any but the simplest digital systems. It is usually impractical for large digital logic systems. The result is that normally all of the distinct timing paths are not exercised on a logic-simulated design, possibly leaving undetected timing errors to cause future problems.

Another problem with the logic simulation approach to timing verification is that it needs to know the values of all the signals in the circuit. This in turn requires either a complete design (including any microcode and programs) to be run on the simulator, or some way of generating value patterns to drive the undefined signals. Waiting until the design is completed to start logic simulation--when this problem is smallest--means that errors are not found until late in the design cycle. Generating patterns to drive

undefined signals in mid-design is a tedious, time-consuming process. To test only the timing of a design, and not its logical correctness as well, requires knowing only when most signals are changing and when they are stable, not their full value behavior.

One of the harder problems in logic simulation for timing verification is how to handle the possible range of propagation delays which a given component may have. There are two basic types of logic simulator systems that are used to address this problem. They are called minimum/maximum-based systems and probability-based systems. The type of system preferable in a particular situation depends on the design methodology used in the design to be verified. The minimum/maximum-based system corresponds to a design methodology where the delay of each component in the system is characterized in terms of a minimum and maximum possible value. These values are added in pairs to yield a pair of values corresponding to the minimum and maximum delays through any given logic path in the design. The probability-based system corresponds to a design methodology in which the delay of each component is given a probability distribution, and these distributions are then combined to determine the delay of a given path through the design, to some pre-specified confidence level.

The next two sections will discuss these two different approaches in greater detail.

1.4.1.1 Minimum/Maximum-Based Logic Simulators

Minimum/maximum-based logic simulators take a minimum and maximum specification for each timing parameter in the system to be simulated which has a possible range of values. TEGAS [Sz72, Sz75], SAGE [Ku76], and LAMP [Ch74, Ch75a] are examples of minimum/maximum-based simulators. These systems have one or more extra states beyond the basic true and false states for specifying that a signal is changing, and that its value is not known. For example, TEGAS when doing precise delay timing uses 6 values: 0, 1, X (initialization value), U (signal rising), D (signal falling), and E (potential spike, hazard, or race). To model a gate with a range of possible propagation delays with this type of logic simulator, the output will be set to these extra values between its minimum and maximum delay. Which value it will be set to depends on the possible behavior of the output for the particular case being simulated.

In general, the minimum/maximum-based system is simpler than the probability-based system, both from the standpoint of the designer and the simulator. It also corresponds to the way that components are normally specified. The problem with the minimum/maximum-based system is that a real design usually could be made to run faster than this system will predict. This is because the probability is quite low that all of the components along a time-critical path will have the maximum or minimum

propagation delay values, if the delays of the components along that path are uncorrelated.

1.4.1.2 Probability-Based Logic Simulators

Probability-based logic simulators are the same as minimum/maximum-based logic simulators, except that they keep track of the mean and variance associated with events in the simulation, instead of the minimum/maximum times associated with an event. The "DIGSIM" system [Ma77a, Ma77b], which uses this approach, assumes that propagation delays are normally distributed, and stores a mean and variance to characterize each delay parameter. When evaluating a gate, it combines the probability function of all of its inputs to come up with a mean and variance characterizing the output of the gate, assuming the time of the output change can be modeled with the normal distribution. It also considers the correlation between the different delay parameters and events within the simulation.

The basic problem with probability-based timing verification systems is that it is difficult to get good data on the distribution of the delays of system components and the timing correlations between the components. IC manufacturers normally test and sort components based on minimum/maximum delays, and not probability distributions. The probability distribution of the delays of components is also a function of the incoming

inspection procedures used. It is also difficult to get good data on the correlations between the delays of the different components being used in the construction of a design. For example, if a set of chips are all produced on one wafer, or in one production run, then their basic propagation delays may all have maximum propagation delays. Components can be mixed from different production runs to minimize this type of problem, but that adds to the manufacturing cost. In a probability-based system, taking into account any correlations is essential to avoid incorrect predictions. Calculations in a probability-based system are also much more difficult for the engineer to perform when checking the results of the simulation, and when determining the number of levels of logic which can be used while doing the initial design.

1.4.2 Worst-Case Path-Searching Algorithms

The worst-case path analysis approach examines all paths through the combinational logic between registers or latches, searching for the longest and shortest paths. In the "GRASP" system [Wo78], which uses this approach, the user identifies starting and terminating points in the design by hand. The system then searches all of the paths between these starting and terminating points to see that they are within their user-specified timing limits. If there are loops in the network that the user hasn't broken with a terminating point, the "GRASP" system proceeds until it reaches some user-specified search limit. It is then up to the user to insert a terminating point in the

loop, and to rerun the analysis.

In the Race Analysis System (RAS) [Ha71], these user-specified starting and ending points for the search are automatically determined by the location of the latches and registers in a design, rather than by hand. The main problem with this approach is that it is unable to take into account the value behavior of the control signals when evaluating the timing of a circuit, and therefore tends to generate numerous irrelevant error messages.

Chapter II

A NEW APPROACH TO TIMING VERIFICATION

A new approach to verifying the satisfaction of timing constraints on large digital systems will now be described. A detailed discussion of a system implemented using this approach will be given in the next chapter. This approach operates on synchronous sequential systems, and checks all of the logic-level timing errors which occur within those systems. These include the non-satisfaction of the set-up, hold, or minimum pulse width time requirements for registers, latches, and other complex functions. In addition to examining for these errors, it checks the timing on control signals which are ANDed with clock signals to verify that they are stable while the clock is asserted, in order to avoid any possible hazard conditions on control-conditioned clock lines. This approach takes into account both the minimum and maximum propagation delays of all of a system's components, including the interconnections between them.

This approach does no low-level circuit analysis, but instead takes as input

parameterized models which define the operation of the gates, registers, and latches. Entirely different techniques are needed to do the low-level circuit analysis, which consider the analog characteristics of the circuits. It also does no system-level timing analysis, which would require it to understand the higher level protocols used between communicating units. The only known ways to do this are either gate-level logic simulation or construction of the system, and then evaluate it in simulated or actual operation. For interconnection delays, a specification of the minimum and maximum delay from the output of one logic element to the input of another logic element is required. The detailed transmission line analysis required to determine the possible range of signal delays of a given interconnection is done in the SCALD Physical Design Subsystem [Mc78a, Mc78b, Sp78, Sp79].

2.1 OVERVIEW OF THE VERIFICATION PROCESS

The timing verification approach developed here simulates one clock period of a circuit, keeping track of when signals can change their value with respect to the clock during that interval. The basic assumption behind such simulation is that signals have a periodic behavior with regard to when they can change their value relative to the central clock, which is normally the case for synchronous sequential circuits.

When a signal can change its value with respect to the clock is in general a

function of the operation being done by the circuit. Calculating when a signal can change for all possible operations is in some cases overly pessimistic, and can cause numerous irrelevant error messages to be generated. In this situation, rather than just simulating one cycle, a number of cycles are simulated. Each simulated cycle is called a *case*, and only considers a subset of the possible operations done by the circuit. This way, each cycle simulated handles the timing properly for the operation being simulated, avoiding irrelevant error messages.

The designer specifies the different cases that need to be simulated individually. The first case is then simulated, detecting any possible timing errors. After that, in going from case-to-case, only the parts of the circuit that are affected by the case analysis are reevaluated. The total number of cycles of the circuit simulated is then equal to the number of cases specified by the designer.

All signals except for the clocks and a few control signals are simulated in terms of whether they are *stable* or *changing*, instead of whether they are *true* or *false*. This *symbolic* timing simulation has the advantage that it tests the circuit for most of the possible state transitions in a signal pass. The resulting savings in computational effort are clearly of factorial (i.e., exponential) order.

To clarify this approach, consider the following example. An edge-triggered register is clocked at a particular time with respect to the central clock. The output of the register can change only during a short time after it is clocked, so that it is guaranteed to be stable for the entire clock period except around the point at which it is

clocked. The output of a gate driven from this register can then be changing only during an interval of time determined by its propagation delay and when its inputs can be changing. If either the output of the register or gate are required to be stable during the part of the circuit cycle when they are possibly changing, then there is a possible timing error.

The first step in the timing verification process is to calculate for each signal in a circuit when it could change during the circuit cycle time. Once this is determined, then it is relatively easy to check all of the timing constraints placed on the circuit. For instance, in order to check the set-up and hold times on a register, all that is required is to determine if its input could be changing at a time when it might be clocked. To check that a control signal which is ANDed with a clock is stable when the clock is asserted is also a straightforward operation.

2.2 CIRCUIT CLOCK PERIOD

Circuits being verified must contain one basic clock, whose period has to be specified. If different parts of the circuit being verified run at different clock rates, then the period specified is the least common multiple of the different clock periods. For example, a processor might have an instruction unit which has a period of 30 nsec and an execution unit which has a period of 15 nsec. In this case, the period specified would

be 30 nsec. Clock signals which occur within the circuit may occur at any time within the clock period.

2.3 TIME UNITS

Time is expressed in two sets of units to the Timing Verifier. When specifying the timing properties of the components in which a design is implemented, absolute time units are used (for example, nanoseconds). When specifying clocks and assertions in the design specification, user-specified *clock units* are employed which are convenient for the designer to use, and which can be scaled with the clock period. For example, the clock units for a design might be defined to be one-eighth of the clock period. This allows the relative timing within the design to automatically scale if the clock rate is slowed down or speeded up as the design is done.

2.4 CIRCUIT MODEL FEATURES

Circuits are described for timing verification purposes in terms of gates, registers, latches, set-up and hold time constraints, and minimum pulse width constraints. More complex functions are then defined in terms of these primitives, through the use of

graphic-based macros, using the SCALD Hardware Description Language [Mc78a, Mc78b, Sp78, Sp79].

The following sections define the value system used to represent the behavior of signals and defines the primitive functions used to specify the design to the Timing Verifier.

2.4.1 Value System Used To Represent Signals

At any instant in time, every signal in the circuit being timing-verified has exactly one of seven values, with the following associated meanings:

Value	Meaning
0	false, or 0
1	true, or 1
S or STABLE	signal is stable, not changing
C or CHANGE	signal may be changing
R or RISE	signal is going from zero to one
F or FALL	signal is going from one to zero
U or UNKNOWN	initial value used for all signals

The value of a signal over the clock period is represented by a linked list, each node of which specifies a signal value and the time duration of that value. The sum of the

durations of all the nodes in the list must exactly equal the period of the circuit being analyzed.

When a signal propagates through a gate or wire where it is delayed by a variable amount of time, then *skew* is added to the signal representation, denoting the uncertainty in when the signal will subsequently change. This skew is maintained separately in the signal representation to preserve information about the width of pulses. This is done to avoid incorrect assertions by the Timing Verifier that minimum pulse width requirements have not been met. If two or more changing signals are combined, the skew of the resulting signal cannot be represented separately. It is therefore incorporated into the signal representation by using the CHANGE, RISE, and FALL values. A detailed example showing this is given in Section 2.8.

2.4.2 Definition of Combinational Functions

This section defines the basic combinational functions used by the Timing Verifier. All other combinational functions may then be defined in terms of these basic functions.

The following tables define the INCLUSIVE-OR (OR), AND (AND), EXCLUSIVE-OR (XOR), CHANGE (CHG), and NOT (NOT) functions for the

seven-value logic system used in the Timing Verifier.

These functions are uniformly defined to give worst-case values. For example, when the signal values "STABLE" and "RISING" are OR'ed together, the resultant signal value given is "RISING". This is because the output in this case will either be stable or a rising edge, and the rising edge is the worst-case value.

A OR B

A ↓ \ B →	0	1	S	C	R	F	U
0	0	1	S	C	R	F	U
1	1	1	1	1	1	1	1
S	S	1	S	C	R	F	U
C	C	1	C	C	C	C	U
R	R	1	R	C	R	C	U
F	F	1	F	C	C	F	U
U	U	1	U	U	U	U	U

A AND B

A ↓ \ B →	0	1	S	C	R	F	U
0	0	0	0	0	0	0	0
1	0	1	S	C	R	F	U
S	0	S	S	C	R	F	U
C	0	C	C	C	C	C	U
R	0	R	R	C	R	C	U
F	0	F	F	C	C	F	U
U	0	U	U	U	U	U	U

A XOR B

A ↓ \ B →	0	1	S	C	R	F	U
0	0	1	S	C	R	F	U
1	1	0	S	C	F	R	U
S	S	S	S	C	C	C	U
C	C	C	C	C	C	C	U
R	R	F	C	C	C	C	U
F	F	R	C	C	C	C	U
U	U	U	U	U	U	U	U

A CHG B

A ↓ \ B →	0	1	S	C	R	F	U
0	S	S	S	C	C	C	U
1	S	S	S	C	C	C	U
S	S	S	S	C	C	C	U
C	C	C	C	C	C	C	U
R	C	C	C	C	C	C	U
F	C	C	C	C	C	C	U
U	U	U	U	U	U	U	U

NOT A

A ↓	
0	1
1	0
S	S
C	C
R	F
F	R
U	U

The output of the "CHANGE" function has the value "UNDEFINED" if any of its inputs are undefined. If all of its inputs are defined, then it has the value "CHANGE" if any of its inputs are changing; otherwise it has the value "STABLE". It is a useful function in modeling complex combinational logic, where the actual function being performed is not significant to the verification process. Common examples are in the modeling of parity trees and adders, in which cases the Timing Verifier cares only

when the outputs of these circuits are changing, not about their actual values. This again results in a factorial-level reduction in the complexity and computational effort of modeling these functions.

2.4.3 Models for Registers and Latches

The Timing Verifier has two models for registers which are shown in Figure 2-1. The first register model just has "CLOCK" and "DATA" inputs, and can change its output only on the rising-edge of its "CLOCK" input. The output of the register will be set to the "CHANGE" state during the time following the rising-edge of "CLOCK" as determined by the minimum and maximum delays of the register. Unless the "DATA" input is a true or false during the rising-edge of the "CLOCK" input, the output will be set to the "STABLE" value for the rest of the cycle; otherwise, it will be set to the value of the "DATA" input. The example in Figure 2-1 shows a minimum delay of 1.0 nsec and a maximum delay of 3.8 nsec being specified for the register, which is 32-bits wide.

The second register shown in Figure 2-1 is the same as the first, except that it has asynchronous "SET" and "RESET" inputs in addition to the "DATA" and "CLOCK" inputs. If the "SET" input is true and the "RESET" input is false, then the output of the register is set to true. If the "RESET" input is true and the "SET" input is false, then the output of the register is set to false. If both the "SET" and "RESET" inputs are

true, then the output is set to "UNDEFINED". If both the "SET" and "RESET" inputs are false, then the register operates identically to a register without the "SET" and "RESET" inputs. For the cases where the "SET" and "RESET" input are changing, the output is set to the "CHANGE" state. If the "SET" and "RESET" inputs are stable, then the output will be stable if the register is not being clocked. The minimum and maximum propagation delays from all of the inputs are the same, and are given by the delay property of the register. If chips with different propagation delays from different inputs are to be modeled, then buffers are used on the various inputs to insert the proper delays. Primitives with different delays from different inputs could be implemented to improve execution efficiency, if desired.

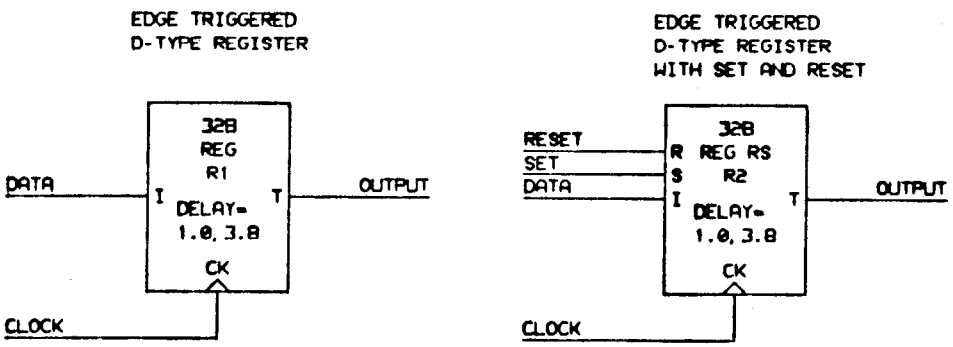


Figure 2-1

Two register models used by Timing Verifier

The Timing Verifier has two models for latches, as shown in Figure 2-2. The

"OUTPUT" of the first latch follows the "DATA" input when the "ENABLE" input is high, and holds the last value given by the "DATA" input when the "ENABLE" input is low. The "SET" and "RESET" inputs on the second latch in Figure 2-2 operate the same as for the register, and override the operation of the latch when they are non-zero. The minimum and maximum propagation delay from all of the inputs on the latch are the same, and is given by the "DELAY" property. For the example shown in the Figure, the minimum propagation delay is 1.0 nsec, and the maximum propagation delay is 3.5 nsec.

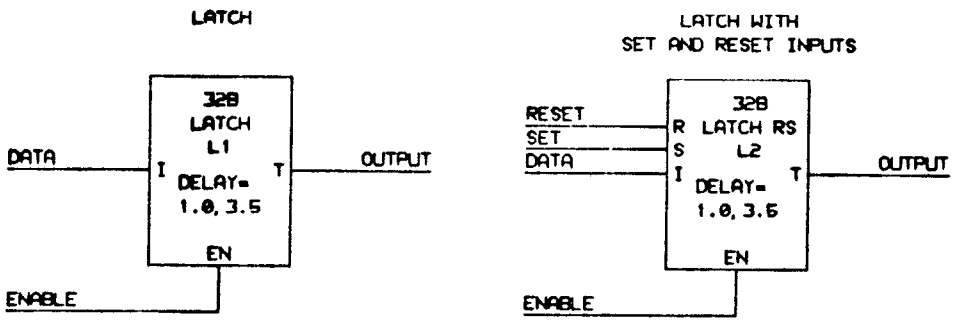


Figure 2-2

Two latch models used by Timing Verifier

2.4.4 Set-up and Hold Time Checkers

There are two primitive functions shown in Figure 2-3 which are used to check set-up and hold times. The first checker is called a "SETUP HOLD CHK", and checks to see that the signal connected to the "I" input is stable for a period around the rising edge of the "CK" input. The "SETUP" property specifies the set-up time interval, which is the length of time the input signal must be stable before the rising edge of the clock input. The "HOLD" property specifies the hold time interval. This is the length of time the input signal must be stable after the rising edge of the clock input.

The second primitive shown in Figure 2-3 is a "SETUP RISE HOLD FALL CHK" primitive. It checks the set-up time interval of the input before the rising edge of the clock input, and the hold time interval after the falling edge of the clock input. It also checks to see that the input "I" is stable for the entire time interval over which the clock input "CK" is true. This type of set-up and hold checker is needed to verify the timing constraints on components such as memory elements.

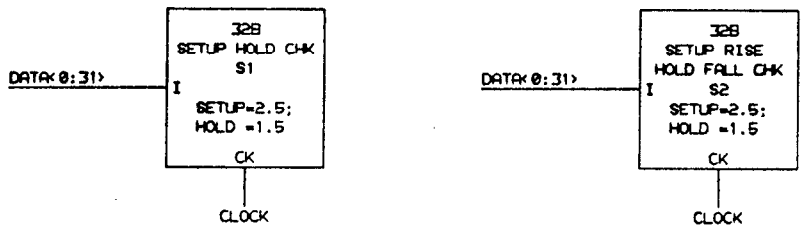


Figure 2-3

Set-up and hold time checkers used by Timing Verifier

2.4.5 Minimum Pulse Width Checking

The minimum pulse width checker primitive is used to specify verification of minimum pulse width constraints. Clock inputs to components typically have a minimum pulse width requirement which says that when they go high, they must stay high for some specified interval of time, and that when they go low, they must stay low for some specified time interval. Figure 2-4 shows the "MIN PULSE WIDTH" primitive, which shows a minimum *high* pulse width of 5.0 nsec being specified, as well as a minimum *low* pulse width of 3.0 nsec.

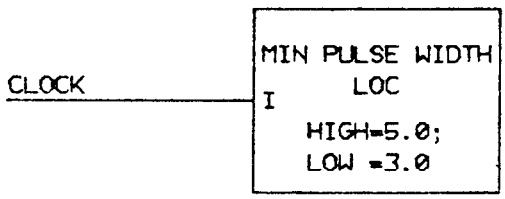


Figure 2-4

Minimum pulse width checker

2.5 SIGNAL ASSERTIONS

In order to be able to analyze partially designed circuits, the Verifier must have timing assertions on as-yet undefined signals. Undefined signals with no assertions are taken to be always stable, to prevent them from giving rise to numerous spurious timing errors. These signals are also put on a special cross reference listing, for appropriate attention from the designer to be directed to them *once*.

For defined signals, two types of assertions are used for specifying clocks, and one

is used for defining the behavior of control and data signals.

2.5.1 Clock Assertions

There are two categories of clock signals: precision and non-precision. The only difference between precision and non-precision clock specifications is the default skew used by the Timing Verifier when none is explicitly given by the designer. Skew is generated by the variation in the interconnection delay to the different parts of a large digital system and by the variations in delay between the different buffers used in the clock generation. In the design of a large digital system, these variations can become quite large, and may degrade performance unacceptably. To reduce such skew to within acceptable limits, the shorter clock paths can have additional delays deliberately inserted into them. Because the delays in a clock distribution system may vary between successive implementations of a design, in many cases it must be adjusted by hand, using some type of adjustable delay for each of the clock lines. By use of this technique, the skew can be reduced to below some designer-specified value. In order to verify the timing in a design which has been so de-skewed, it is necessary to describe in detail how the clocks will be adjusted within the design specification. A number of features have been provided to make this task as easy as possible, and will be described in the section on evaluation directives.

If a clock signal is adjusted to some specified skew, then an assertion can be given within its signal name signifying that fact. Assertions are given at the end of signal names and are preceded by a period. They are considered part of the signal name by the rest of the SCALD system, which thereby guarantees that all of the assertions for a given signal are consistent by definition.

The format for the assertions for the precision and non-precision clocks are

```
<precision clock>       ::= <signal name> .P <assert spec>
<non-precision clock>   ::= <signal name> .C <assert spec>
<assert spec>           ::= <value specification>
                            <skew specification> <polarity assertion>
<value specification>   ::= <time range> |
                            <time range> , <value specification>
<time range>            ::= <time> | <time> - <time> | <time> + <time>
<time>                  ::= <real number>
<skew specification>    ::= | ( <minus skew> , <plus skew> )
<minus skew>            ::= - <real number> | <zero>
<plus skew>             ::= <real number> | <zero>
<time>                  ::= <clock units>
<clock units>           ::= <real number>
<polarity assertion>    ::= | L
```

An example of clock specification is

XYZ .C4-6 L

which states that the clock signal goes from high to low at time 4, and from low to high at time 6. The signal

XYZ .C2-3,5-6

is high from 2 to 3 and from 5 to 6, and is low for the rest of the clock cycle. If a single time is given instead of a range, a time interval of one clock unit is assumed. For example,

XYZ .C2,5

is equivalent to the previous signal. The signal

XYZ .P2,5

is again equivalent, except that it is a precision clock, which means that it has a different default skew. In general, it was found in the design of the S-1 Mark IIA processor that having two types of clocks -- those that have been adjusted to reduce skew, and those that haven't -- was convenient. The motivation was to only adjust those clocks which must be adjusted, in order to reduce the aggregate cost of clock de-skewing.

If a plus sign is given between the two time variables instead of the minus sign, then the second number specifies a width in nanoseconds, rather than the time of the end of the pulse in clock units. This allows widths of clocks to be specified which don't scale with the cycle-time of the circuit. For example,

specifies a clock that goes high at clock unit time 2, and stays high for 10.0 nsec thereafter.

2.5.2 Stable Assertions

The stable assertion is used to specify when a control or data signal is stable, and when it may be changing. Its general form is

SIGNAL NAME .S <value specification> <polarity assertion>

For example, the name XYZ .S4-8 says that the signal is stable from time 4 to time 8, and that it may be changing during the rest of the cycle.

This type of assertion has several uses. First, it allows the designer to specify his assumptions about when signals are valid (i.e., not changing) as he creates them in the design process, and those assumptions will be used by the Timing Verifier until the signals are generated by hardware. For signals so generated, the designer's initial timing assertion is checked against the timing of the actual signal, and an error is given if the assertion is violated. Having these assertions on signals greatly improves the readability

of the design, since a signal name explicitly includes a specification of when it is valid.

Putting these "stable" assertions on interface signals is the key to the ability to verify a design in sections. After each section is verified, SCALD checks to see that all interface signals have the same timing assertions on them. If no section of a design being verified has a timing error and if all of the interface signals of all such sections have consistent assertions on them, then the entire design must be free of timing errors. This modular verification capability is in turn crucial to the real-world utility of the timing verification approach described here, just as the use of subroutines and procedures is to structured programming.

2.5.3 Interconnection Delay Specification

Taking into account the effects of interconnection delays throughout the design process is essential if maximum system performance is to be attained when the design is completed. The consideration of these delays needs to be approached from two different points of view, depending on whether or not the design is far enough along to allow the actual interconnection delays to be calculated. If the interconnection delays can be calculated from detailed simulation of the transmission line properties of the interconnections in the circuit-as-packaged, then these delay values are used by the Timing Verifier when checking timing constraints within the design. If the

interconnection delays are not yet known, the Timing Verifier uses a default interconnection delay for each signal. If the designer wishes, he may specify within the design a range for the interconnection delay for a specific signal, which will then override the default specification.

2.6 EVALUATION DIRECTIVES

Evaluation directives are used to specify:

- That the control signals being ANDed with a given clock signal must be stable while the clock is asserted. This is used to detect possible hazards which could be generated on the output of a gate, resulting in false clocking of the circuit that the gate controls. Section 1.3.2 gives an example of this type of timing error. In addition, these directives cause the Timing Verifier to assume that the control signals will be enabling the gate, so that its output value will be determined only from the value behavior of the clock signal.

- The tuning of clocks in systems that have hand-adjusted clocks to reduce skew. Additional information is needed here since the prints don't specify how the clocks are adjusted.

Consider the circuit shown in Figure 2-5. The clock signal "CK .P2-3 L" is being ANDed with the control signal "WRITE .S0-6 L" to generate a write-enable pulse for the RAM array. The "&H" directive specifies checking that the control signal "WRITE .S0-6 L" is stable during the interval over which the clock is asserted, to ensure that the "write" will be either solidly enabled or completely disabled. In addition, this directive says the timing specified by the clock signal is to be adjusted so that it refers to the time at which the *output*, rather than the *input*, of the gate changes. This corresponds to a circuit in which the clock signals are adjusted to eliminate the skew generated by gating of the clock lines. The "&H" directive also specifies the assumption that the value of the "WRITE .S0-6 L" signal is enabling the gate, allowing the clock signal to always propagate through the gate.

The different evaluation directive and their meaning are:

E	Evaluate gate with no special action. This is the default mode.
W	Zero wire going into gate.
Z	Zero gate and wire going into it.
A	Check to see that other inputs to gate are not changing when this input is asserted (true). In calculating the output of the gate, assume that the other inputs are enabling the gate.
H	This directive has the combined effects of the "Z" and "A" directives.

For example, in Figure 2-5 the "&Z" directive on the signal "CK .P0-4" states that the clock timing refers to the time at which the *output* of the gate changes. If multiple directives are given after a signal, such as "&HZ", then the first letter refers to the first

level of gating after the directive, the second refers to the second level of gating, etc. There is no limit on the length of a directive string.

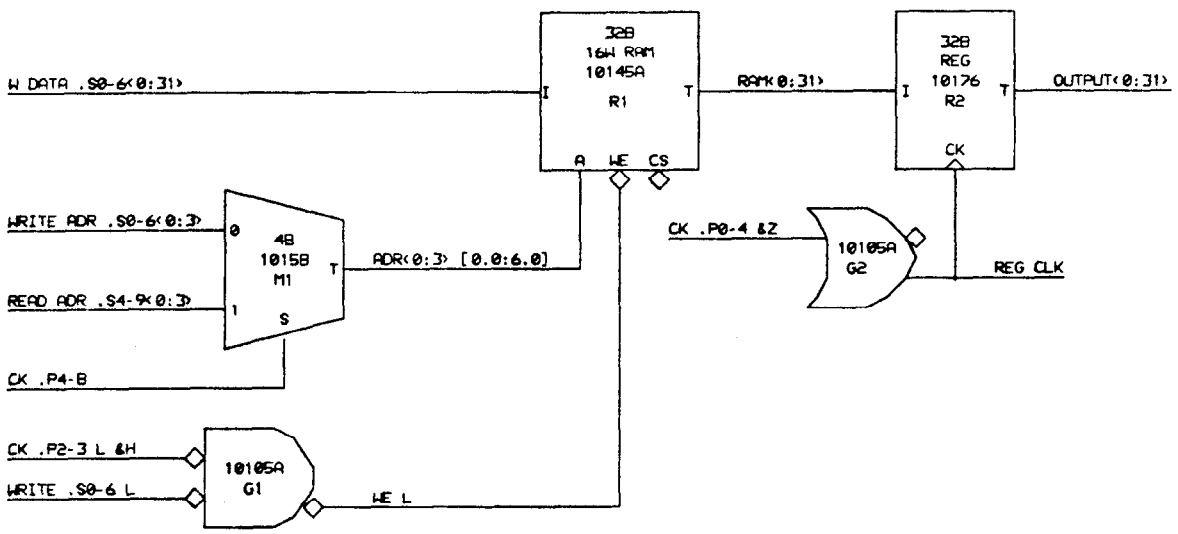


Figure 2-5

Example macro definition

2.7 CASE ANALYSIS

When the timing verification of all possible operations of a circuit are reduced to the simulation of one cycle of the circuit through the use of the "STABLE" and "CHANGING" values, overly pessimistic results are sometimes generated. When this occurs, the timing verification can be broken down into a number of separate simulations of the circuit. Each simulation tests out distinct operations of the circuit which place different timing constraints on the circuit. In doing these separate simulations, only those parts of the circuit that are affected by the case analysis are reevaluated, permitting most case analysis to be done quite efficiently.

Some circuits have paths through them which are never used, and which require the case analysis feature to avoid generating pessimistic timing delays. Consider the example shown in Figure 2-6. If the circuit is analyzed without case analysis, where the signal "CONTROL SIGNAL" has the value "STABLE", then the delay from the signal "INPUT" to the signal "OUTPUT" would be calculated to be 40 nsec. The problem is that the Timing Verifier would be unable to determine that both of the multiplexers could not select the "1" input at the same time. To use case analysis, the designer would specify that the signal "CONTROL SIGNAL" needs to analyzed separately for the cases when it is true and when it is false. For the first case, the Timing Verifier would then

set the signal "CONTROL SIGNAL" to the value "0" whenever the circuit would normally set it to the value "STABLE". For the next case, it would set it to the value "1" whenever the circuit would normally set it to the value "STABLE". By doing this, the two select lines on the multiplexers would always be set to complementary values, and the delay from the signal "INPUT" to the signal "OUTPUT" would be calculated to be 30 nsec for both cases.

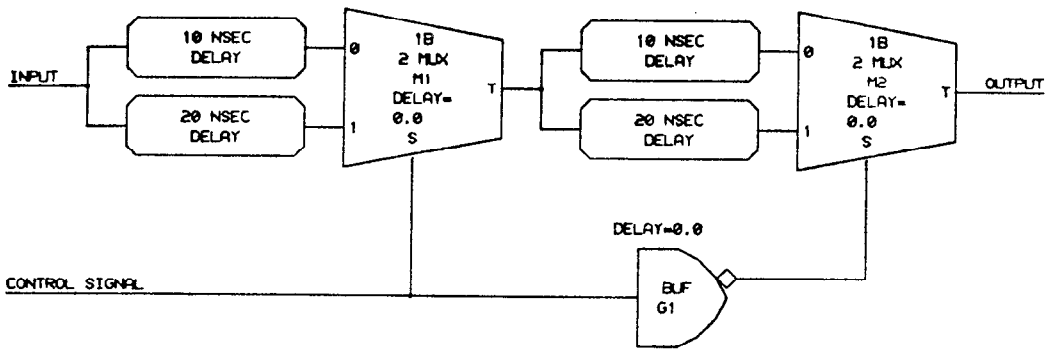


Figure 2-6

Example of circuit requiring case analysis

2.7.1 Case Specification

The designer must identify and specify the cases which need to be handled by case analysis. He does this by mapping the "STABLE" states into either "0" or "1" for the signals which control the operation that the circuit is to perform. Consider the following specification:

```
CONTROL SIGNAL=0;
CONTROL SIGNAL=1;
```

This specification gives two cases to be evaluated for the circuit in Figure 2-6. The first case causes the circuit to be simulated with the signal "CONTROL SIGNAL" having its "STABLE" values mapped into the value "1". The second case specified causes the circuit to be simulated again with the signal "CONTROL SIGNAL" having its "STABLE" values mapped into "0".

2.8 REPRESENTATION OF SIGNAL VALUES

The Timing Verifier represents in memory the value of each signal over the circuit cycle time. It uses a linked list, which has the format shown in Figure 2-7. For each signal, there is a "VALUE BASE" record with a free storage link, a field to store the skew, a pointer to the evaluation string, and a pointer to the linked list representing the signal value. The "VALUE" record specifies the signal value and the width of that value. The sum of all of the "VALUE WIDTH" fields on the linked list is required to exactly equal the cycle time of the circuit being verified, for consistency-checking purposes and to avoid ambiguity.

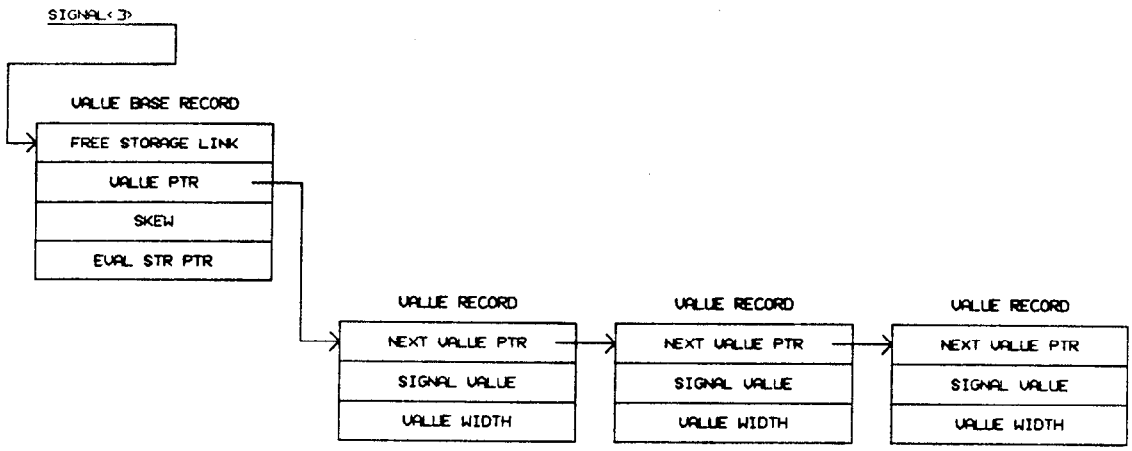


Figure 2-7

Data structures used to represent signal values

The "SKEW" field is used to represent skew caused by delaying the signal by a variable amount of time. Consider the example in Figure 2-8. The gate has a minimum delay of 5.0 nsec and a maximum delay of 10.0 nsec. The two input signals will be ORed together as if the gate had zero delay, and the value of the output signal will then be delayed by the minimum delay. The skew field will then be set to the difference between the maximum and the minimum delay of the gate. By doing this, rather than by using "RISING" and "FALLING" values to represent the uncertaintity in when the signal will transition between a zero and a one, the symmetry information

about the width of pulses is preserved since the rising and trailing edges of the signal are delayed by the same amount. When modeling a technology in which the rising and falling values of signals are different, this algorithm will have to be modified to take such asymmetry into account.

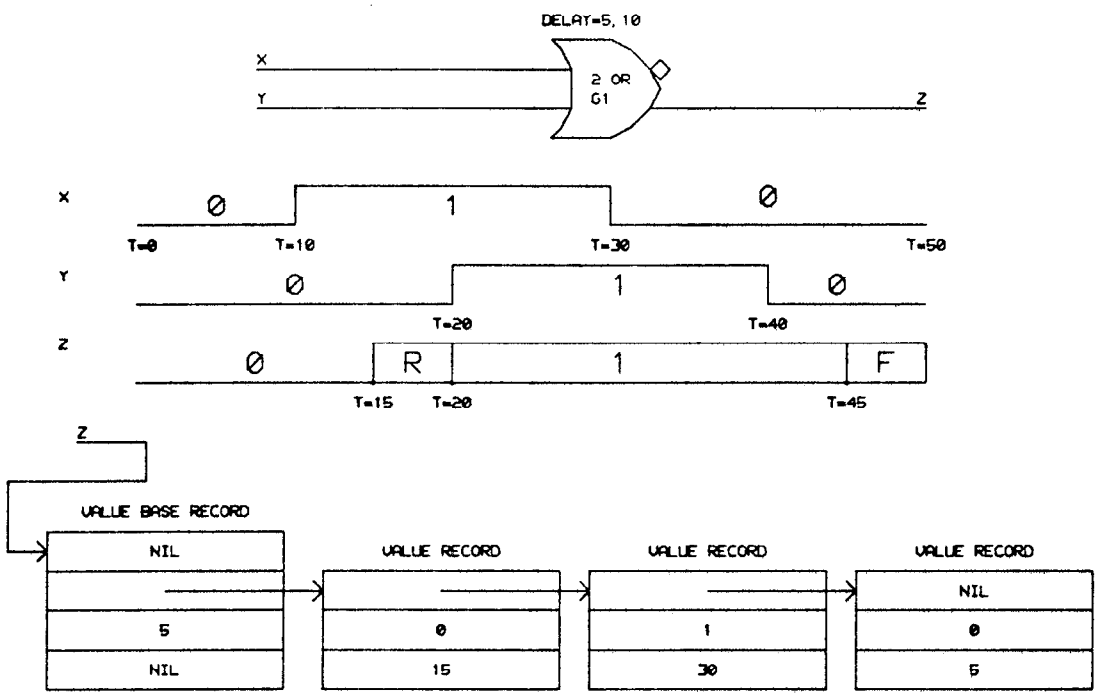


Figure 2-8

Example showing how skew is handled

This separate representation of skew can be used in essentially any situation in which a signal value is merely being delayed by a variable amount. However, if two signals are being combined, then the skew of the combined value in general cannot be simply represented with a single field. Because of this, when two signals are combined, their skew is inserted into the resultant signal representation using the "RISING" and "FALLING" values. For example, the output signal "Z" from the last example is shown in Figure 2-9 with its skew inserted into the signal value.

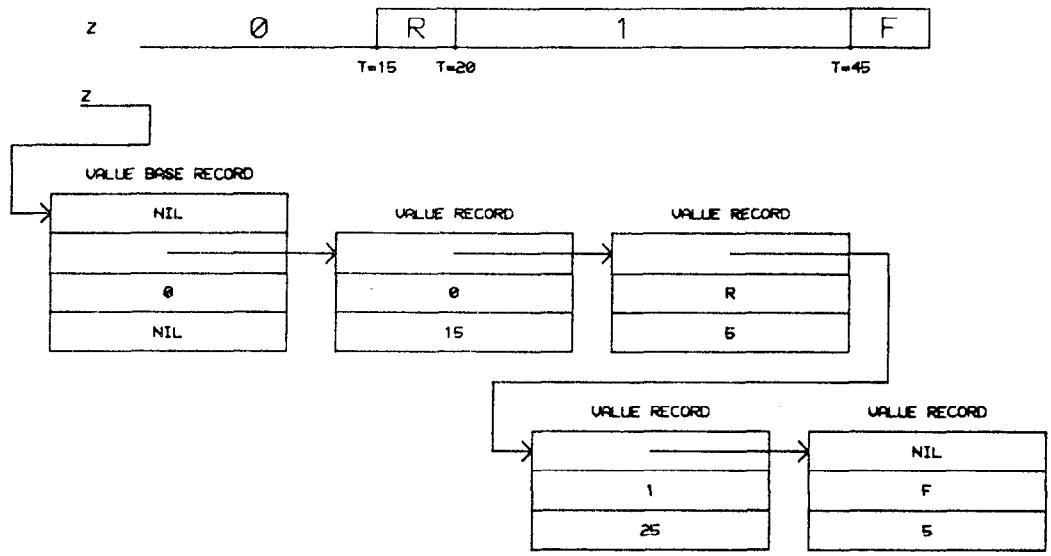


Figure 2-9

Output signal Z with skew represented in signal value

The "EVAL STR PTR" field is used to keep track of the evaluation string associated with the signal value. For example, if the evaluation string "HZZW" is given on the input of a gate, then each letter specifies how to evaluate a subsequent level of

gating. Each gate will remove the letter which specifies how to evaluate it, and will pass along the rest of the string and the output value from the gate, to specify how to evaluate the next level of gating. The string "HZZW" would then specify the evaluation of four levels of gating, with the "H" controlling the first level, and the "W" controlling the fourth and last level.

2.9 TECHNIQUE USED FOR CIRCUIT EVALUATION

The first step in evaluating a circuit is to initialize to "UNDEFINED" all signals without assertions. Signals with clock assertions are set to the value specified. Signals with stable assertions are set to the value "STABLE" during the time specified by the assertion, and to the value "CHANGING" the rest of the time. Signals which are specified in the case analysis file will be set to the value specified for the case being calculated whenever they otherwise would be given the value "STABLE".

In the next step, the Timing Verifier evaluates all of the primitives which define the circuit by looking at their current input values and, based on these, calculating new output values. Whenever a new output value is different from its old value, all of the primitives which are driven by that output are added to a list of primitives to be evaluated during the next pass of the Timing Verifier. This process continues, reevaluating those primitives which have had their inputs changed, until all of the

signals stop changing. At that point, the Timing Verifier knows the value of each signal over the clock period, for the first case to be analyzed.

The next step is to evaluate all of the set-up and hold times, and minimum pulse width checkers, based on the value of their inputs, and to output error messages reporting any errors detected. This error checking includes set-up and hold time constraints specified both by the set-up and hold time primitives and by the "&A" and "&H" evaluation directives.

At this point, the first case has been evaluated, and the Timing Verifier is ready to evaluate the next case. This involves changing the values of those signals specified by the case analysis file, and reevaluating those primitives whose inputs are affected. This process is continued, as in the first case, until all signals stop changing. At that point, the second case has been checked. The Timing Verifier will continue this process, incrementally reevaluating the network, until all of the cases specified by the designer have been checked.

Chapter III

APPLICATION OF TIMING VERIFICATION

This chapter gives a set of examples of the use of the SCALD Timing Verifier, and discusses statistics on its use.

3.1 SPECIFICATION OF TIMING PROPERTIES OF COMPONENTS

The timing properties of the components which constitute a design are described to the Timing Verifier in terms of a set of built-in primitive functions. These functions include *gates, registers, latches, multiplexers* and *set-up/hold/minimum-pulse-width checkers.* The definitions of these primitives are given in Section 2.4.

A manufacturer's specification of a 16-word by 4-bit register file chip is given in

Figure 3-1 to 3-4. The specification to the Timing Verifier of this component's timing properties is given in Figure 3-5, and is expressed in the SCALD Hardware Description Language [Sp79]. A brief description of the basic features of this SCALD language will be sketched in the following example.

Figure 3-5 represents a macro to be expanded every time the chip with name "16W RAM 10145A" is used. This definition defines a memory whose width in bits is given by the variable "SIZE", which is defined when the macro is called. A call to this macro is shown in Figure 2-5, showing a size of 32 bits being specified. The "PARAMETER" body in Figure 3-5 defines the parameters which can be passed to it, and also specifies the number of bits which each parameter may be passed. For example, the parameter declaration "I<0:SIZE-1>" says that the "I" parameter has bits numbered from "0" to "SIZE-1," and is thus "SIZE" bits wide. The string "/P" after a signal name says that the signal is a parameter of the macro, and is used as a consistency check, as well as improving the readability of the macro. The string "/M" says that the signal is local to the macro. If neither "/M" or "/P" are given, then the signal is global. The name of this macro "16W RAM 10145A" is given in the lower center portion of the drawing.

The "16W RAM 10145A" definition checks the set-up and hold time constraints on the input signals "I<0:SIZE-1>", "CS", and "A<0:3>", by using the "SETUP HOLD CHK" and "SETUP RISE HOLD FALL CHK" primitives. For example, the upper "SETUP HOLD CHK" body checks that the "I<0:SIZE-1>" inputs are stable at least 4.5 nsec before the falling edge of the write-enable ("WE") pulse and for at least -1.0 nsec

after it. The leading "-" on the signal "- WE" says to use the complement of the signal "WE". The "SETUP RISE HOLD FALL CHK" body, in the lower left hand corner, checks that the address lines of the memory ("A<0:3>") are stable at least 3.5 nsec before the rising edge of the write-enable pulse, that they are stable while the write-enable pulse is high, and that they stay stable for at least 1.0 nsec after the falling edge of the write-enable pulse. The "MIN PULSE WIDTH" primitive checks that whenever the write-enable pulse goes high, it does so for at least 4.0 nsec.

The "CHG" and "3 CHG" gates at the top of the page cause their outputs to change after the delay specified by the "DELAY" parameter whenever their inputs change. For the "CHG" gate, the "DELAY" parameter says that it has a minimum delay of 1.5 nsec, and a maximum delay of 3.0 nsec. For the "3 CHG" gate, the "DELAY" parameter says that it has a minimum delay of 3.0 nsec, and a maximum delay of 6.0 nsec.

F10145A • F10545A

16 x 4 REGISTER FILE (RAM)

F10K-VOLTAGE COMPENSATED ECL

GENERAL DESCRIPTION – The F10145A and F10545A are high-speed 64-bit Random Access Memories organized as a 16-word by 4-bit array. External logic requirements are minimized by internal address decoding, while memory expansion and data bussing are facilitated by the output disabling features of the Chip Select ($\overline{CS}$) and Write Enable ($\overline{WE}$) inputs.

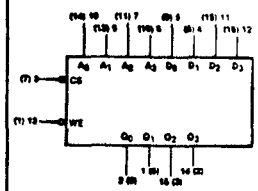
A HIGH signal on $\overline{CS}$ prevents read and write operations and forces the outputs to the LOW state. When $\overline{CS}$ is LOW, the $\overline{WE}$ input controls chip operations. A HIGH signal on $\overline{WE}$ disables the Data input (D_n) buffers and enables readout form the memory location determined by the Address (A_n) inputs. A LOW signal on $\overline{WE}$ forces the Q_n outputs LOW and allows data on the D_n inputs to be stored in the addressed location. Data exits in the same logical sense as presented at the data inputs, *i.e.*, the memory is non-inverting.

- READ ACCESS TIME—7 ns TYP
- 50 kΩ INPUT PULL-DOWN RESISTORS
- OUTPUTS CAN BE WIRED-OR FOR EASY MEMORY EXPANSION
- CHIP SELECT ACCESS TIME—4 ns TYP
- VOLTAGE COMPENSATED, INSENSITIVE TO POWER SUPPLY VARIATIONS
- FULLY COMPATIBLE WITH ALL 10,000 SERIES ECL

PIN NAMES

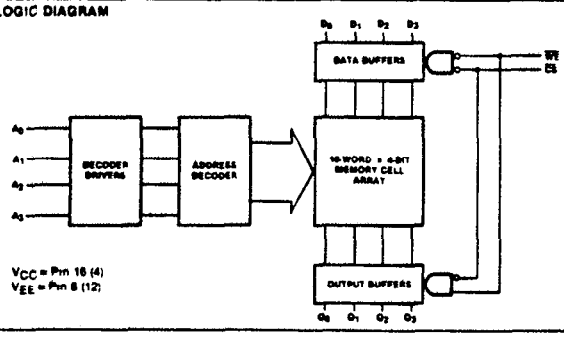
$\overline{CS}$	Chip Select
A_n	Address Lines
D_n	Data Input Lines
$\overline{WE}$	Write Enable
Q_n	Data Output Lines

LOGIC SYMBOL



V_{CC} = Pin 16 (4)
V_{EE} = Pin 8 (12)
( ) = Flatpak

LOGIC DIAGRAM



V_{CC} = Pin 16 (4)
V_{EE} = Pin 8 (12)

CONNECTION DIAGRAM
DIP (TOP VIEW)
PACKAGE OUTLINE 6B

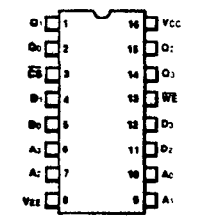


Figure 3-1

Manufacturer's data sheet for register file chip

Reprinted with permission from Fairchild Camera and Instrument Corporation, 464 Ellis Street, Mountain View, Ca. 94042.

DC CHARACTERISTICS: V_{EE} = −5.2 V, V_{CC} = GND

SYMBOL	CHARACTERISTIC	LIMITS B	LIMITS TYP	LIMITS A	UNITS	T_A	CONDITIONS
I_{IH}	Input Current HIGH $\overline{CS}$, A0 — A3 $\overline{WE}$, D0 — D3			 200 220	μA	25°C	$V_{IN} = V_{IHA}$
I_{EE}	Supply Current	−150	−100		mA	25°C	Inputs and Outputs Open

AC CHARACTERISTICS: V_{EE} = −5.2 V, T_A = 25°C

SYMBOL	CHARACTERISTIC	LIMITS B	LIMITS TYP	LIMITS A	UNITS	CONDITIONS
	Access/Recovery Times					
t_{ACS}	Chip Select Access	3.0	4.5	6.0	ns	Figures 1, 3
t_{RCS}	Chip Select Recovery	3.0	4.5	6.0	ns	
t_{AA}	Address Access	4.5	6.5	9.0	ns	
	Write Times					
	Set-Up					
t_{WSD}	Data	4.5	3.0		ns	
t_{WSCS}	Chip Select	4.5	2.5		ns	
t_{WSA}	Address	3.5	1.5		ns	Figures 1, 2a
	Hold					
t_{WHD}	Data	−1.0	−2.5		ns	
t_{WHCS}	Chip Select	0.5	0.0		ns	
t_{WHA}	Address	1.0	−1.0		ns	
t_{WR}	Write Recovery Time	3.0	4.5	6.0	ns	Figures 1, 3
t_{WS}	Write Disable Time	3.0	4.5	6.0	ns	
t_w	Write Pulse Width, Min	4.0	2.5		ns	Figures 1, 2a
t_{CS}	Chip Select Pulse Width, Min	4.0	2.5		ns	
	Select Times					
	Set-Up					
t_{CSD}	Data	4.5	3.0		ns	
t_{CSW}	Write Enable	4.5	2.5		ns	
t_{CSA}	Address	3.5	1.5		ns	Figures 1, 2b
	Hold					
t_{CHD}	Data	−1.0	−2.5		ns	
t_{CHW}	Write Enable	0.5	0.0		ns	
t_{CHA}	Address	1.0	−1.0		ns	
	Transition Times					
t_{TLH}	20% to 80%	1.5	2.5	3.9	ns	Figures 1, 3
t_{THL}	80% to 20%	1.5	2.5	3.9	ns	

Figure 3-2

Manufacturer's data sheet for register file chip

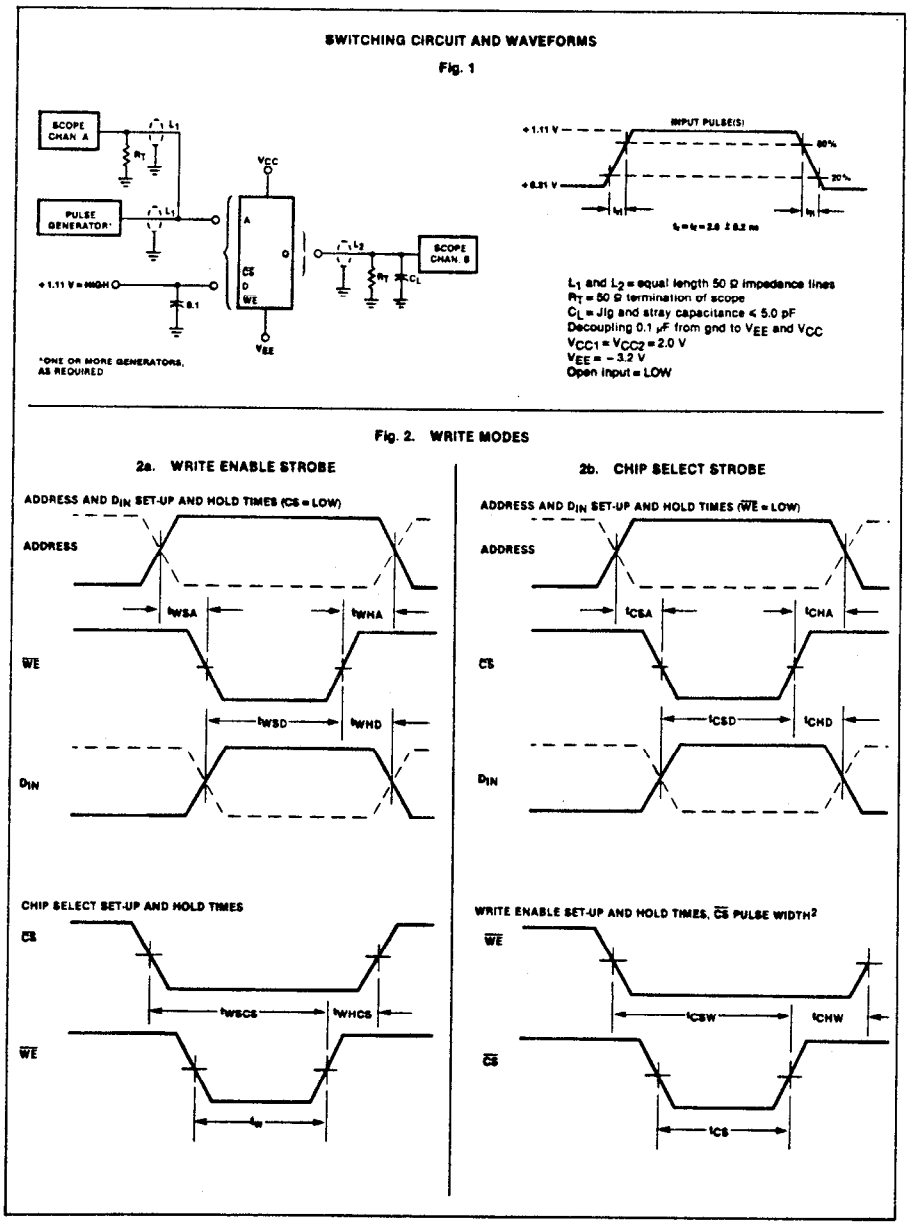


Figure 3-3

Manufacturer's data sheet for register file chip

Reprinted with permission from Fairchild Camera and Instrument Corporation, 464 Ellis Street, Mountain View, Ca. 94042.

WAVEFORMS (Cont'd)

Fig. 3. READ MODES

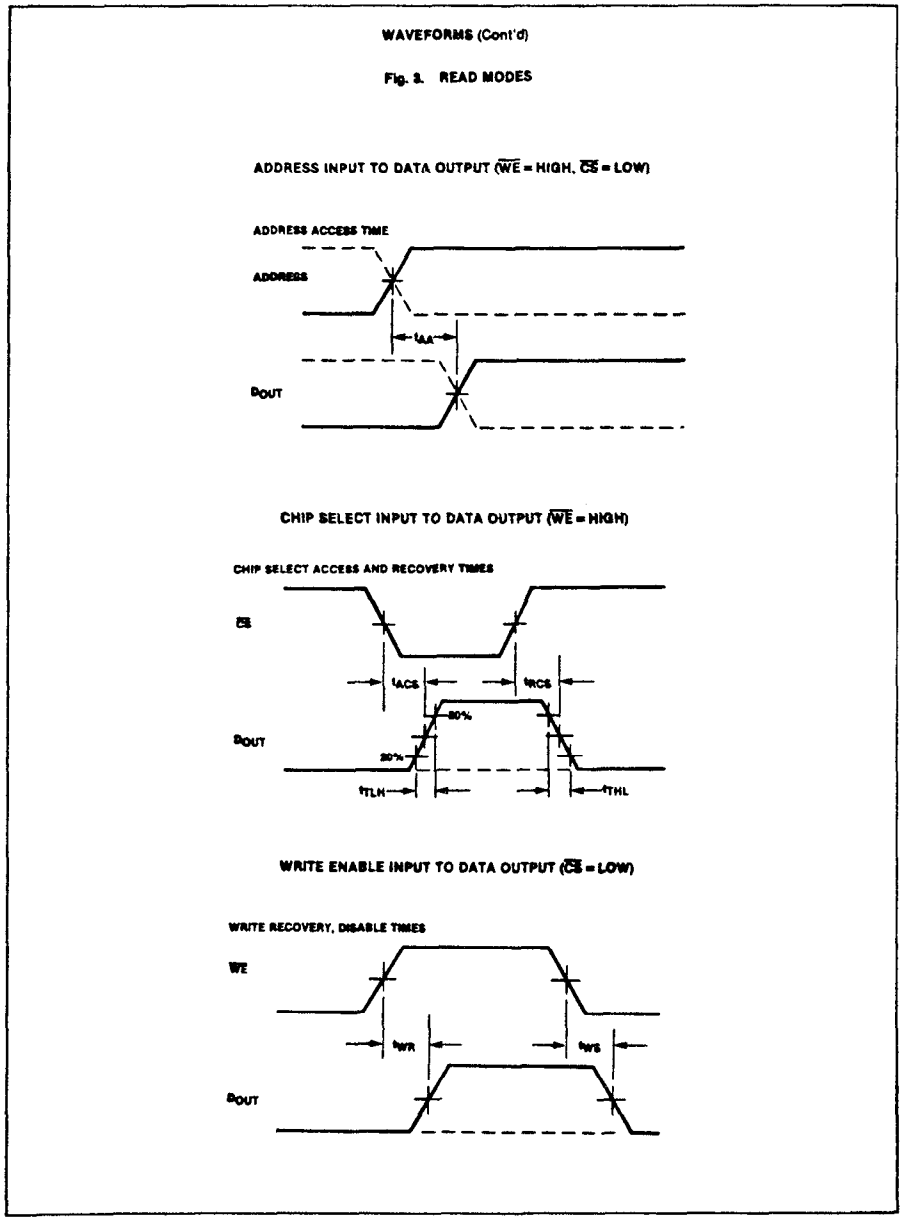


Figure 3-4

Manufacturer's data sheet for register file chip

Reprinted with permission from Fairchild Camera and Instrument Corporation, 464 Ellis Street, Mountain View, Ca. 94042.

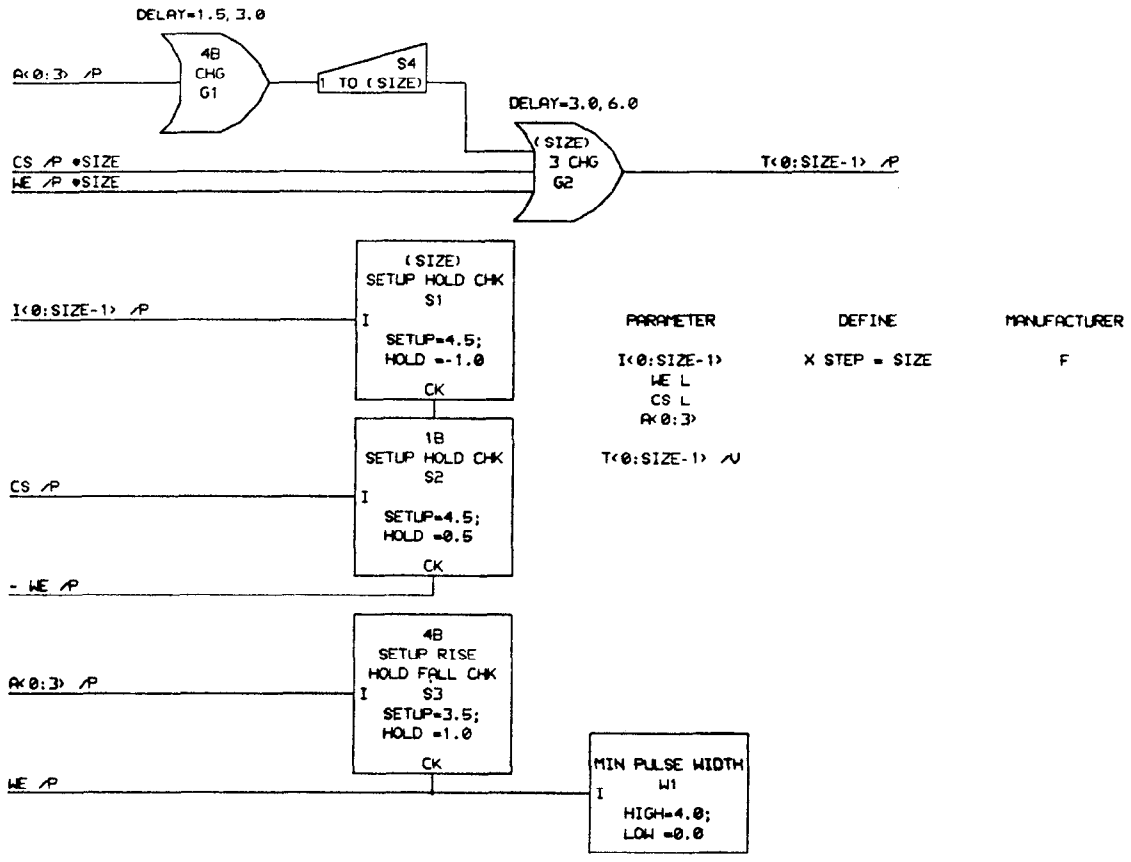
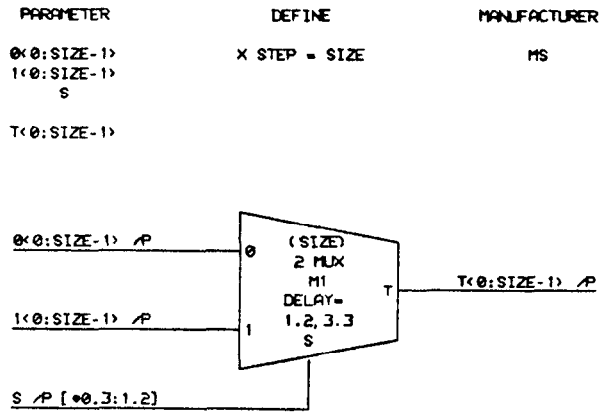


Figure 3-5

Definition of a 16-word random access memory chip

The definition of a 2-input multiplexer chip is given in Figure 3-6. This definition is given in terms of the "2 MUX" primitive, which has a minimum specified

delay of 1.2 nsec, and a maximum delay of 3.3 nsec from any of the inputs to the output. The select input ("S") has an additional minimum delay of 0.3 nsec, and an additional maximum delay of 1.2 nsec, which is added to the delay of the "2 MUX" primitive.



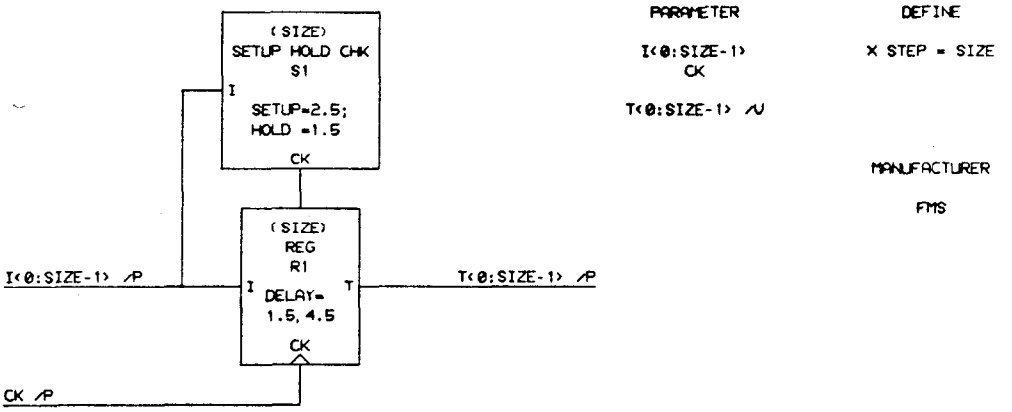
10158

Figure 3-6

Definition of a 2-input multiplexer chip

The definition of an edge-triggered register is given in Figure 3-7. This

definition is stated in terms of the register primitive. It has a minimum delay of 1.5 nsec and a maximum delay of 4.5 nsec. The "SETUP HOLD CHK" primitive specifies a set-up time of 2.5 nsec and a hold time of 1.5 nsec for the data input "I<0:SIZE-1>" with respect to the "CK".

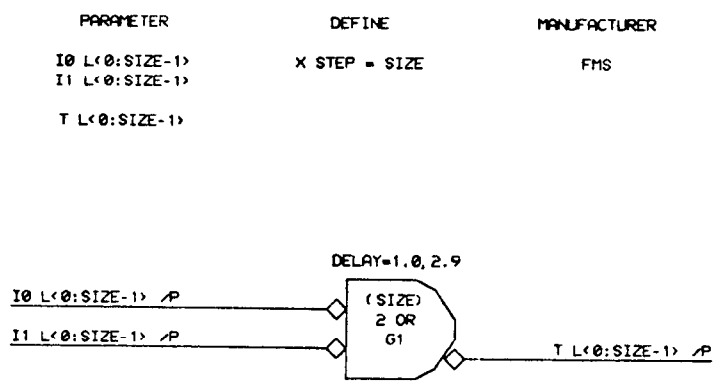


10176

Figure 3-7

Definition of an edge-triggered register chip

The definition of a 2-input OR gate is shown in Figure 3-8. It is defined in terms of the "2 OR" primitive. The gate has a minimum propagation delay of 1.0 nsec and a maximum propagation delay of 2.9 nsec.

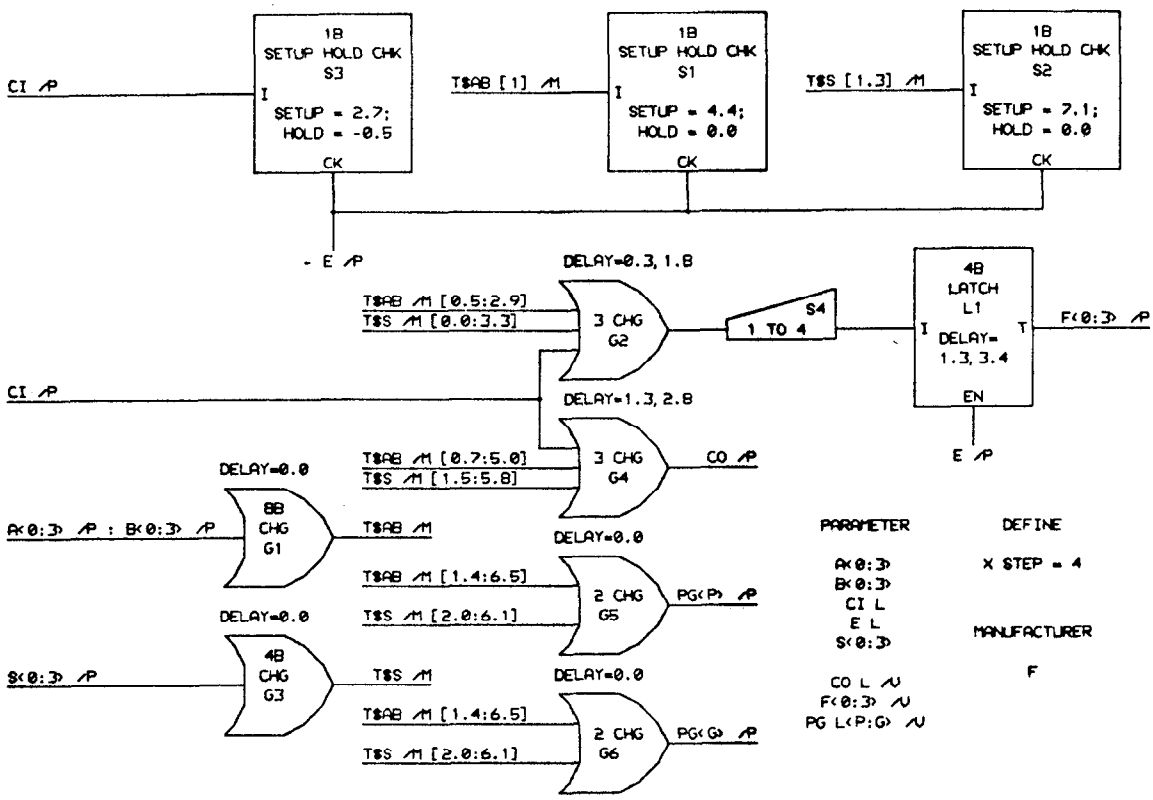


10105A

Figure 3-8

Definition of a 2-input OR gate

The Timing Verifier definition of an arithmetic/logic unit with output latch is shown in Figure 3-9. This unit performs one of 16 functions on the three data inputs "A<0:3>", "B<0:3>", and "CI". The function to perform is selected by the input "S<0:3>". The input "E" enables the output latch. The "SETUP HOLD CHK" primitives check the set-up and hold time constraints on the data inputs when the latch is closed. The propagation delay from the data inputs to the output is specified by the group of "CHG" gates.



100181

Figure 3-9

Definition of an arithmetic/logic chip (ALU)

3.2 CIRCUIT VERIFICATION EXAMPLE

Figure 2-5 shows an circuit example to be analysed by the Timing Verifier. This circuit consists of a 16-word by 32-bit register file, a 32-bit output register, a 2-input multiplexer which selects between the read and write addresses for the register file, and several gates. The circuit is designed to run with a cycle time of 50 nsec. The minimum/maximum pair of default wire delays used by the Timing Verifier in checking this circuit was 0.0/2.0 nsec, and the default clock skew for the clocks was - 1.0 to +1.0 nsec. The time unit used in the specification of the clocks and assertions is 6.25 nsec, which gives 8 clock units per cycle.

One of the most useful features of the Timing Verifier is its ability to analyse all of the timing properties of a design *as the design proceeds*, rather than having to wait to be used until the design is completed. As such, it can accept as input the description of this circuit example, which would typically be a small section of a much larger system, and determine if it contains any timing errors.

The stable assertions on the input signals which are not generated in this circuit are crucial to the ability to verify a design in sections. For example, the assertion on the signal "W DATA .S0-6" states that it is stable from time 0 to time 6, and that it may be changing during the rest of the cycle, i.e. from time 6 to time 8. The assertion on the

signal "READ ADR .S4-9" says that it is stable from time 4 to time 9, and may be changing the rest of the cycle, i.e. from time 1 to time 4. This may seem somewhat unusual at first, but the cycle time of the circuit is 8 clock units long, and the assertion specification is taken to be modulo the cycle time.

Considering interconnection delays on incomplete designs presents some interesting problems. If the actual wire delays are known for the signals in the circuit, they can be used to do the analysis. If not, the Timing Verifier will use a default wire delay, unless the designer specifies wire delays for specific signals. The minimum/maximum default wire delay pair of 0.0/2.0 nsec was used for all of the wires in this example, except for the address lines on the register file, where the designer specified that it could be anywhere from 0.0 to 6.0 nsec.

Figure 3-10 exhibits the summary output listing generated by the Timing Verifier, showing the values of the signals over the cycle time of the circuit. For example, the first entry says that the address lines "ADR<0:3>" are stable at the beginning of the cycle, and that they start changing 0.5 nsec into the cycle. They then go stable 5.5 nsec into the cycle, and stay stable until 25.5 nsec into the cycle. They are then changing from 25.5 nsec to 30.5 nsec, after which point they stay stable for the rest of the cycle.

Figure 3-11 contains the set-up and hold time errors which were detected by the Timing Verifier. The first message states that the "SETUP HOLD CHK" primitive specified a set-up time interval of 3.5 nsec, followed by a hold time of 1.0 nsec, and that the set-up time was violated. The next two lines give the values seen by the "SETUP

HOLD CHK" primitive on the data and clock inputs. They show the data not becoming stable until 11.5 nsec into the cycle, and the clock starting to rise 11.5 nsec into the cycle. Thus, the set-up time interval specified was missed by the full 3.5 nsec. The next error message shows that the set-up time interval on the output register was violated. The data didn't go stable until 47.5 nsec into the cycle and the clock starts rising at 49.0 nsec, thereby missing the specified set-up time interval of 2.5 nsec by 1.0 nsec.

```
Values of all signals

ADR<0:3> . . . . . S:0.0, C:0.5, S:5.5, C:25.5, S:30.5
CK .P0-4 . . . . . R:0.0, 1:1.0, F:24.0, 0:26.0, R:49.0  (constant value)
CK .P2-3 . . . . . 0:0.0, R:11.5, 1:13.5, F:17.8, 0:19.8  (constant value)
CK .P4-8 . . . . . F:0.0, 0:1.0, R:24.0, 1:26.0, F:49.0  (constant value)
OUTPUT<0:31> . . . . . S:0.0, C:0.5, S:7.5
RAM<0:31>. . . . . S:0.0, C:5.0, S:20.5, C:30.0, S:45.5
READ ADR .S4-9<0:3>. . . . . S:0.0, C:6.3, S:25.0
REG CLK . . . . . R:0.0, 1:1.0, F:24.0, 0:26.0, R:49.0
W DATA .S0-6<0:31> . . . . . S:0.0, C:37.5
WE . . . . . 0:0.0, R:11.5, 1:13.5, F:17.8, 0:19.8
WRITE .S0-6 . . . . . S:0.0, C:37.5
WRITE ADR .S0-6<0:3> . . . . . S:0.0, C:37.5
```

Figure 3-10

Timing Verifier output showing values of signals

```
Setup, Hold and Minimum Pulse Width errors ....

Setup time error: Setup Time = 3.5, Hold Time = 1.0
CK INPUT   = WE                                  (+0.0)      0:0.0, R:11.5, 1:15.5, F:17.8, 0:21.8
DATA INPUT = ADR                                 (+0.0)      S:0.0, C:0.5, S:11.5, C:25.5, S:36.5

Setup time error: Setup Time = 2.5, Hold Time = 1.5
CK INPUT   = REG CLK                             (+0.0)      R:0.0, 1:3.0, F:24.0, 0:28.0, R:49.0
DATA INPUT = RAM                                 (+0.0)      S:0.0, C:5.0, S:22.5, C:30.0, S:47.5
```

Figure 3-11

Set-up and hold time errors found by Timing Verifier

3.3 PROCESSOR DESIGN TIMING VERIFICATION

The SCALD Timing Verifier has been used in the design of the S-1 Mark IIA processor [SP79]. This exercise has served to validate the utility of the described approach to timing verification, and has also provided performance statistics. The Mark IIA is a highly pipelined processor which is designed to issue a new instruction every 50 nsec. The machine has a vector instruction unit which is being designed to process vector operands at a pipeline rate of one every 25 nsec. The design rules employed by the Timing Verifier in the examination of the Mark IIA design are:

- The overall circuit cycle time was specified to be 50 nsec. Major parts of the design operate at a 25 nsec cycle time.

- The minimum/maximum default interconnection delay pair was 0.0/2.0 nsec, and was used except for those signals for which the designer specified a different range. The actual interconnection delays for the Mark IIA design based on the transmission line properties have not yet been calculated via detailed simulation, and as such, cannot be checked. Refined rules for future designs could take into account the number of loads on a run, and the size of the different loads. It is easy to vary the rule that is used, but more difficult to find a single rule which works well in practice in all instances. The other constraint is that it is convenient to the point of necessity to have a rule which is easy for the designer to use in estimating the delay while doing the design. A rule which only slightly improves the accuracy of the delay, but which is difficult for the designer to readily employ, is clearly not worthwhile.

- The precision clocks are assumed to have a skew of +1.0 to -1.0 nsec from the times specified, and the non-precision clocks are assumed to have a skew of +5.0 to -5.0 nsec. The implementated design will have a set of programmable delay lines which will be used to trim the clocks' skews to this specification.

- The propagation delay for the integrated circuits used are the minimum and maximum delay specifications given by the manufacturer. Where a part is

manufactured by a number of different companies, the worst-case delay numbers, determined from the minimum and maximum values of the different specifications from the various companies are used.

The next two sections will discuss the execution statistics derived and the evaluation made as results of using the Timing Verifier to examine the S-1 Mark IIA design.

3.3.1 Design Experience in Using the Timing Verifier

The basic approach used to validate the functioning of the Timing Verifier and to assess its utility has been to extensively exercise it on the design of a high performance digital processor, the S-1 Mark IIA. It has been used frequently to check the design of this system for timing behavior as it has progressed toward implementation. The approach taken has been to advance the design for about a day, and then to enter the new design into the SCALD system, via the Stanford University Drawing System (SUDS) [He72] running on the S-1 Mark I system operating in PDP-10 simulation mode. The design is then processed through the SCALD Macro Expander, which checks the design for syntax errors and generates a file which represents the expanded design. The expanded design is then read into the Timing Verifier, which checks all of the timing constraints imposed on it.

This daily introduction into the design effort of feedback about timing errors has been exceedingly helpful. It has allowed possible timing errors to be corrected while the associated design is fresh in the minds of the designers, and before a great deal of additional logic is designed which depends on the timing properties of the logic already designed.

A typical circuit from the S-1 Mark IIA design is shown in Figure 3-12. It consists of a 36-bit arithmetic/logic unit with output latch, a 36-bit debugging/status register with load-enable, and a function decoder that controls the function select input to the arithmetic/logic unit. All of the inputs and outputs from this circuit contain assertions which specify when they can change. This allows the timing of this circuit to be checked, either by itself or with the rest of the design. Adding these stable assertions to the interface signals greatly adds to the readability of the design, making its timing features exceedingly clear.

The timing constraints that need to be checked by the Timing Verifier for the circuit in Figure 3-12 are the set-up, hold, and minimum pulse width constraints on the output latch of the arithmetic logic unit and the debugging/status register. In order to do this the Timing Verifier has to calculate when the inputs to these functions can change, relative to their clocks, and the width of the clocks.

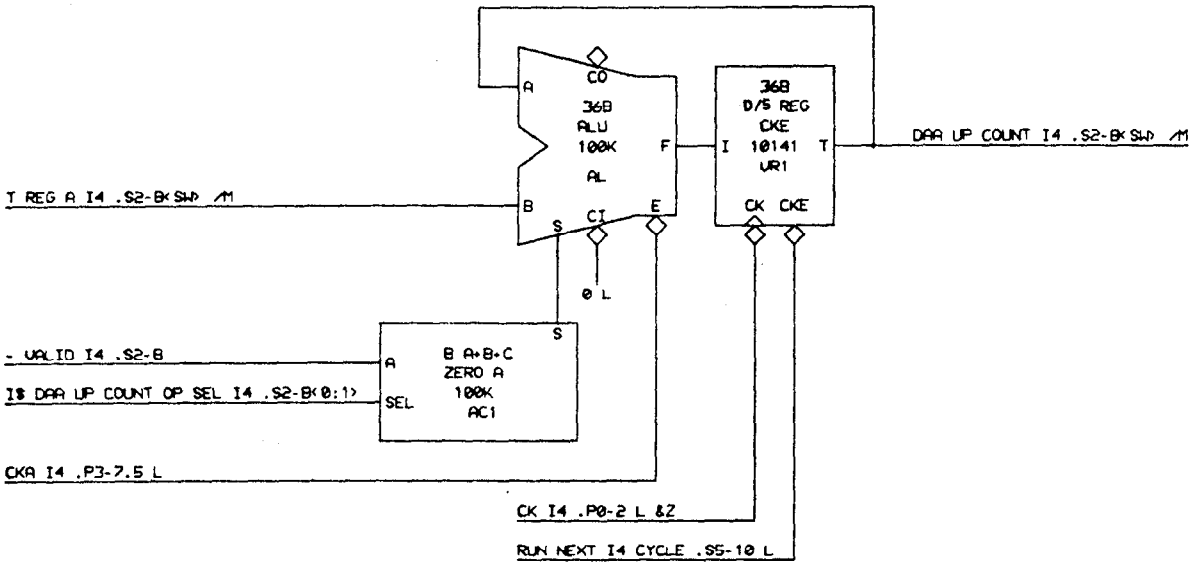


Figure 3-12

Typical arithmetic circuit in the S-1 Mark IIA design

3.3.2 Execution Statistics of Timing Verifier

The basic execution statistics associated with running the Macro Expander on the S-1 Mark I system [Sp78] (which has a throughput rate approximately equivalent to an IBM 370/168) for a major portion of the Mark IIA design consisting of 6357 MSI ECL-10K and ECL-100K chips are shown in Table 3-1; this portion contains approximately 97,709 2-input gates-equivalent of logic and 1,803,136 bits of memory. These execution statistics are resolved into three portions of this processing task. The first part is that involved in reading the input files and building the data structures to represent the design. Next, the Macro Expander does an expansion of the design to generate a summary listing, and builds up a data structure which resolves all synomyms between different signals; this is Pass 1. Finally, the Macro Expander expands the design again, this time outputting the fully elaborated design for use by the Timing Verifier or the SCALD Layout Program; this is Pass 2.

MACRO EXPANSION EXECUTION STATISTICS	Elapsed Time, minutes
Reading input files and building data structures	1.92
Pass 1 of Macro Expansion	8.42
Pass 2 of Macro Expansion	6.18
	16.52
TIMING VERIFIER EXECUTION STATISTICS	
Reading input files and building data structures	4.45
Generating cross reference listings	0.72
Verifying circuit	6.75
Generating timing summary listing	0.22
	12.14
Total for both Timing Verifier and Macro Expander:	28.66

Table 3-1
Execution statistics for 6357 chip design example

The Macro Expander generates a design representation in terms of primitive definitions which are built into the Timing Verifier. Table 3-2 gives a listing of the primitive types generated for the circuit example processed by the Macro Expander. There were a total of 22 primitive types used, and each type was used 376 times on the average, for a total of 8282 primitives. This gives 1.3 primitives per chip in the design. These chips are MSI components and RAMs which average about 20 gates per chip for the logic, and 1K bits per RAM. The reason why so few primitives were used is because the primitive types, such as registers ("REG RS") and multiplexers ("8 MUX"), are reasonably high-level ones, and each primitive represents an arbitrarily wide data path.

For example, the average width of a primitive was 6.5 bits. If this symmetry had not been exploited, then 53,833 rather than 8,282 primitives would have been used to represent the circuit.

PRIMITIVE TYPE	NUMBER GENERATED	AVERAGE NUMBER OF BITS WIDE
2 AND	374	6.1
2 CHG	438	1.7
2 MUX	288	7.2
2 OR	917	2.3
3 CHG	152	1.3
3 OR	430	2.5
4 CHG	6	7.8
4 MUX	83	10.9
4 OR	167	3.8
5 OR	241	1.3
8 CHG	112	4.0
8 MUX	51	6.8
BUF	1507	4.8
CHG	693	8.5
LATCH	209	6.5
LATCH RS	102	8.7
MIN PULSE WIDTH	361	1.0
REG	246	25.4
REG RS	21	3.7
SETUP HOLD	1010	9.7
SETUP RISE HOLD FALL	343	17.7
XOR	531	2.0
	8282	6.5

Table 3-2
Primitive definitions generated for 6357 chip example

The execution statistics for the Timing Verifier are shown in Table 3-1. The Timing Verifier took 4.45 minutes reading in the output from the SCALD Macro Expander, and then building up its data structures. It then generated cross reference listings, which aid the designer in finding where signals are used within the design. The

next step was the timing verification process itself. This required 6.75 minutes, or about 49 milliseconds per primitive. In doing this verification, 20,052 events were processed, where an event was caused by an output being given a new value, which in turn caused all primitives which use that output value to be updated. An event then took 20 milliseconds to process. This verification was for a single case.

The amount of time required to analyze an additional case is proportional to the number of events which have to be processed for that case. In general, only those signals which are affected by the case analysis need to be recalculated. The Mark IIA processor is a pipelined processor, in which every pipeline stage must take the same amount of time to execute. It was found that case analysis was only rarely required for that design. However, for some design styles, e.g. those in which variable length cycles are used, case analysis is essential.

Table 3-3 gives the storage required for data structures used during the Timing Verification. Representing the circuit description is the single largest part of this requirement, representing 37.8%. The circuit description is comprised of a characterization of each primitive used, with a description of which signals were passed to each of its parameters. This is the main data structure used while the circuit is being verified, and its average size is 260 bytes per primitive. The S-1 Mark I PASCAL compiler doesn't pack its records, so all fields require four bytes, except characters and booleans, which take one byte.

STORAGE TYPE	K BYTES	% OF STORAGE
CIRCUIT DESCRIPTION	2149	37.8%
SIGNAL VALUES	1843	32.4%
SIGNAL NAMES	660	11.6%
STRINGS SPACE	600	10.6%
CALL LIST ARRAY	389	6.9%
MISCELLANEOUS	41	0.7%
	5684	100.0%

Table 3-3
Storage required by Timing Verifier for 6357 chip example

The next largest part of the storage requirement is for the storage of signal values. A linked list is stored for each signal in the system representing its value. For the current example, there were 33,152 of these value lists stored, each of which had a base record followed by an average of 2.97 value records. The average amount of memory needed to store the value of a given signal was then 56 bytes. The storage area for keeping track of signal names is used to point to the value definition for each bit of a signal vector, and to record which primitives define and use a given signal; it required 11.6% of the total storage used. The string space, which stores the text strings used by the other data structures, accounts for 10.6% of the storage space. The "CALL LIST ARRAY" describes which primitives need to be reevaluated when a given bit of a signal is updated, and accounts for 6.9% of the storage space. The "MISCELLANEOUS" category represents a number of minor data structures used within the Timing Verifier, which represent 0.7% of the storage. The Timing Verifier program consists of 4700 lines of PASCAL code and requires 214K bytes of memory when loaded with run-time

support.

In general, a compiler that packed the records to take up minimum space would permit a significant reduction in Timing Verifier storage requirements. Also, additional programming to optimize the data structure for space could result in a non-negligible storage saving. The approach taken for this research was to get a system up and running relatively quickly, to evaluate the basic concepts of this approach, and not to try and produce an optimized implementation for use in a production environment. Even so, this system has been sufficiently efficient and powerful to be used extensively in the design of the S-1 Mark IIA processor.

Chapter IV

CONCLUSIONS AND FUTURE RESEARCH

4.1 CONTRIBUTIONS

This thesis has developed an algorithm and associated implementation that:

- Verifies all of the timing constraints in synchronous sequential circuits, including those containing value-dependent timing.

- Verifies timing constraints in these circuits as the design proceeds, without the need for microcode or diagnostic programs, by doing most of the verification in a value-independent fashion.

- Allows large digital logic circuits to be conveniently verified in sections through the use of assertions on interface signals.

Many large digital circuits designed today are synchronous sequential digital systems. Previous approaches used to detect timing errors in these designs have been unable to handle the portion of circuits for which the timing is a function of the values of the control signals (e.g., path-searching systems), or have generally been necessarily incomplete in their testing for all possible timing errors (e.g., gate-level logic simulation.)

Path-searching systems search for the longest (or most critical) path between two registers or latches. These systems have the fundamental limitation that they cannot simulate the portions of the circuit which need to know the value behavior of some of the signals in order to determine the timing of the circuit. Therefore, the handling of clocks used in unusual ways, such as driving the select line of a multiplexer, or the treating of circuits requiring case analysis tend to result in large numbers of spurious error messages being generated. Some of these systems generate so many irrelevant error messages that they have been found to be inconvenient to use.

Gate-level logic simulation simulates a digital logic system, taking into account the timing properties of the components. Simulation is generally an inefficient way to check for timing errors, because of the need to simulate a large number of cycles of the operation of the circuit in order to test all of the different state transitions which must be checked; only by doing so can the designer be certain that all of the worst-case paths through the design have been tested. Such logic simulators are also inconvenient to use,

because the complete value behavior of each signal in the design needs to be available, whereas to check only the timing behavior typically requires much less information. This additional information required to simulate a design comes in the form of microcode and diagnostic programs and data patterns to drive signals not defined by the circuit; these are normally laborious to generate. It is particularly inconvenient to generate and update such data sets on many occasions, as the design progresses, which in turn is necessary to allow timing constraints to be verified as the design evolves. With Timing Verifier usage, it is convenient to check the design on a regular, frequent basis, so that timing errors can be found as soon as they occur.

This thesis has developed and evaluated in realistic use an algorithm to verify all of the logic level timing constraints in the design of synchronous sequential digital systems, in a way that eliminates the basic problems of previously available methods. This algorithm is computationally efficient, requiring an amount of time per case to be analysed of the same order as what a logic simulator would require to simulate only one micro-cycle of the circuit. It is also convenient to use and eliminates the need to generate microcode, diagnostics, and data patterns to drive signals not yet generated, by verifying most of the design in a value-independent fashion. Moreover, it allows a design to be conveniently checked for timing errors as it proceeds, and in a highly modular fashion. A Timing Verifier has been implemented using this algorithm and used in the design of a high performance pipelined processor, the S-1 Mark IIA. Extensive use of the Timing Verifier in the design of this processor has shown it to be a convenient and highly effective tool.

The early detection of timing errors in designs can result in a significant reduction in the design time required to make a digital logic system run at a given speed, in addition to supporting creation of faster-running implementations of designs so realized. In the design of large, high-speed digital logic systems, the handling of timing errors and the optimization of the design for timing constitutes a large fraction of the total system development time. The timing verification technique described here can substatially improve the procedures which are currently being used, thereby making a significant impact on how digital computing systems are designed and implemented.

4.2 FUTURE RESEARCH

There are a number of areas of research in this field to be addressed in the future. These include verifying the timing of asynchronous circuits, taking into account different rising and falling delays, consideration of correlations between different events within the circuit, and evaluating a system that does probability-based analysis, instead of the minimum/maximum-based analysis used in the present work.

4.2.1 Asynchronous, Self-Timed Circuits

One form of asynchronous circuit currently being discussed in the literature is the "self-timed" circuit [Me80, Se79]. These are circuits in which each module within a design keeps track of how long it takes to compute a result. Modules then do hand-shaking between themselves, keeping each other from proceeding until all of their inputs are valid. One of the advantages of this type of design is that a central clock doesn't have to be distributed, which is a current problem in VLSI designs where only one level of metallization is used (as clock lines carried significant distances in "slow", non-metallic lines develop unacceptably large skew. A number of manufacturers are looking at multiple layers of metallization as another possibility.) The verification technique developed here could be used to determine the delay of the basic modules, to determine how much of a delay needs to be inserted in the circuit which specifies when the module is "done". Checking the hand-shaking logic between the different modules is a functional verification correctness-checking problem which is beyound the scope of this thesis.

4.2.2 Different Rising and Falling Delays

When designing with an implementation technology such as nMOS, in which there are greatly differing rising and falling delays, it is overly pessimistic to just use the longer of the two delays, as is done in the timing verification technique developed here. The fundamental problem is that, except for clock circuitry, the Timing Verifier doesn't know the value of a given signal, and therefore doesn't know whether to use the rising or falling delay value. Now, in all cases except for multiple inverting levels of logic, merely using the maximum of the rising and falling delays is the correct choice. One approach is to recognize multiple inverting levels of logic, and to automatically adjust the delays specified for those gates to take into account the different rising and falling delays. This approach would allow the Timing Verifier to continue checking the timing in a value-independent fashion, while taking into account the different rising and falling delays.

4.2.3 Correlations Within Digital Systems

Another limitation of the Timing Verifier is that it doesn't consider possible correlations in the circuit being checked. Figure 4-1 shows a circuit in which an edge-triggered register is loaded from either its old output value or from some new input value, depending on the value of its select line. This circuit also has a buffer on its clock line which inserts a relatively large amount of skew into the register clock. The minimum delay of the register and the multiplexer together are longer than the hold time of the register, but the Timing Verifier checks the hold time on the register from the end of the rising edge, and then calculates when the output of the register could be changing, starting from the rising edge of the clock pulse. In doing this, the Timing Verifier thinks that the input data to the register is changing during the hold time for the register, and it generates a false error message. The problem is that the Timing Verifier presently does all of its calculations in terms of absolute time values, and ignores information about the relative timing of when the register is clocked and when the input data can change. Thus there is a *correlation* between the two signals which are inputs into the "SETUP HOLD CHK" primitive which the Timing Verifier should consider; since it currently does not, it occasionally emits false timing errors in such circumstances.

Correlation-engendered false timing error tends to be a problem in counters, shift registers, and other circuits in which there is feedback from the output of a register into

its inputs. The approach which has been taken is to make the designer explicitly insert a "fictitious" delay into the feedback path which is at least as long as the skew on the clock signal. This delay is inserted with a text macro called "CORR" to make it clear what the designer is trying to do. Figure 4-2 shows this delay inserted. It suppresses generation of the false error message, while allowing other possible errors associated with this circuit to be checked. This approach has worked out well in the S-1 Mark IIA processor design, but puts a significant burden on the designer. It would be preferable if a simple method could be devised to automatically solve this problem. A logic simulation system called "DIGSIM" [Ma77a, Ma77b] has been implemented which keeps track of the relation of different events to each other and which therefore handles this type of situation correctly. The techniques used there could be incorporated into a system based on the concepts developed in this thesis. The extra complexity and memory required was determined to make this not feasible for the S-1 Mark IIA design work, and was not implemented in the current version of the Timing Verifier.

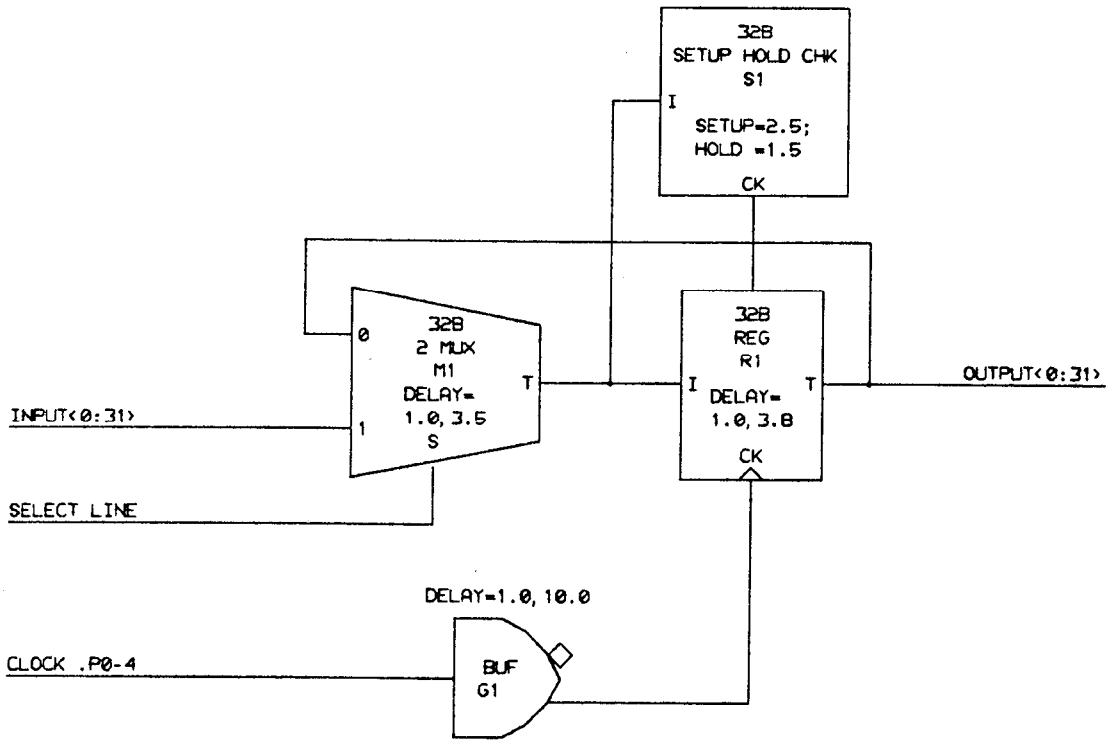


Figure 4-1

Example showing correlation problem

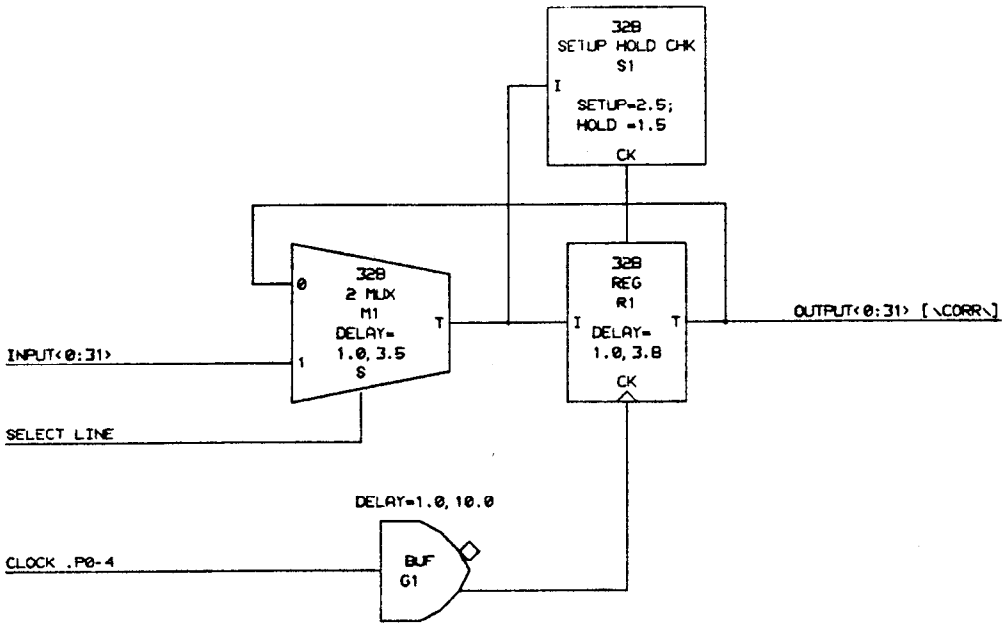


Figure 4-2

Example showing correlation problem with extra delay inserted

4.2.4 Probability-Based Analysis

The Timing Verifier does minimum/maximum-based analysis. This means that all propagation delays are specified with a minimum and maximum possible value. The design is then checked to see that it will always perform properly if all components perform within their specified time ranges.

Probability-based analysis allows a distribution to be specified for each propagation delay. The design is then checked to see that all of the paths in it are within their required limits with a specified level of probability. The idea is that there is a low probability of *all* of the components along a given path having either of their extreme values. The "DIGSIM" logic simulator does this type of analysis and assumes that all the components' timing characteristics have normal distributions [Ma77a, Ma77b]. The type of analysis the timing verification should use depends on the technology being studied and the design techniques a given design team wishes to use. For example, if all of the components along a given path come from that same production run, then their delays may be quite highly correlated, and the probability-based analysis may therefore indicate that a circuit built with them will work with a probability that will be much higher (or lower) than will be found in the real production environment. The problem here is that the probability-based analysis assumed that the delays between the different components were uncorrelated. Another problem with the probability-based analysis arises when the manufacturer starts testing the timing properties of the components and sorting them into faster and slower groups. All of the fast components may be sold to one customer, spoiling an orginal normal distribution for the rest of the customers. Another problem is that the actual distribution of delays for a given component may vary significantly from manufacturer to manufacturer, and even from month to month from the same one. All the manufacturer guarantees is the minimum and maximum timing specifications, which are easy to measure. The present approach may therefore be the best one in such circumstances.

For those technologies and environments in which probability-based design is appropriate, an equivalent of Timing Verifier capability is nonetheless needed. Such a Timing Verifier could keep track of means and variances, rather than minimum and maximum values.

Chapter V

REFERENCES

[Ba78] Bayegan, H.M., and Aas, E., "An Integrated System For Interactive Editing of Schematics, Logic Simulation and PCB Layout Design," Proc. 15th Design Automation Conference, Las Vegas, Nev., June 1978, 1-8.

[Bo71] Booth, T.L., Digital Networks and Computer Systems, John Wiley and Sons, Inc., 1971.

[Bo77] Bose, A.K., and Szygenda, S.A., "Detection of Static and Dynamic Hazards in Logic Nets," Proc. 14th Design Automation Conference, New Orleans, Louisiana, June 1977, 220-224.

[Br72a] Breuer, M.A., "A Note on Three Valued Logic Simulation," IEEE Trans. on Computers, April 1972, 399-402.

[Br72b] Breuer, M.A., Design Automation of Digital Systems, New Jersey: Prentice-Hall, 1972.

[Br76] Breuer, M.A. and Friedman, A.D., Diagnosis and Reliable Design of Digital Systems, Computer Science Press, Inc., 1976.

[Ch72] Chaney, T.J., Ornstein, S.M., and Littlefield, W.M., "Beware the Synchronizer," COMPCON-72 IEEE Computer Society Conference, San

Francisco, Ca., September 1972.

[Ch74] Chang, H.Y., Smith, G.W. Jr., and Schmidt, L.D., "LAMP: System Description," The Bell System Technical Journal, Vol. 53, No. 8, October 1974, 1431-1449.

[Ch75a] Chappell, S.G., Elmendorf, C.H., and Schmidt, L.D., "LAMP: Logic-Circuit Simulators," The Bell System Technical Journal, Vol. 53, No. 8, October 1975, 1451-1475.

[Ch75b] Chawla, B.R., Gummel, H.K., and Kozak, P., " MOTIS--An MOS Timing Simulator," IEEE Transactions on Circuits and Systems, Vol. CAS-22, No. 12, December 1975, 901-910.

[Ch76] Chicoix, C., Pedoussat, J., and Giambiasi, N., "An Accurate Time Delay Model For Large Digital Network Simulation," Proc. 13th Design Automation Conference, San Francisco, Ca., June 1976, 54-60.

[Co69] Couranz, G.R., "An Analysis of Binary Circuits Under Marginal Triggering Conditions," Technical Report #15, Washington University Computer Systems Laboratory, St. Louis, Mo., November 1969.

[Ev77] Evangelisti, C.J., Goertzel, G., and Ofek, H., "Designing with LCD: Language For Computer Design," Proc. 14th Design Automation Conference, New Orleans, Louisiana, June 1977, 369-376.

[Ev78] Evans, D.J., "Accurate Simulation of Flip-Flop Timing Characteristics," Proc. 15th Design Automation Conference, Las Vegas, Nev., June 1978, 398-404.

[Ha69] Hays, G.G., "Computer-Aided Design: Simulation of Digital Design Logic," IEEE Trans. on Computers, Vol. C-18, January 1969, 1-10.

[Ha71] Harrison, R.A. and Olson, D.J., "Race Analysis of Digital Systems Without Logic Simulation," Proc. 8th Design Automation Workshop, Atlantic City, New Jersey, June 1971, 82-94.

[He72] Helliwell, D., "The Stanford University Drawing System," Stanford Artifical Intelligence Laboratory, Stanford University, 1972.

[Ho75] Hoehne, H., and Piloty, R., "Design Verification at the Register Transfer Language Level," IEEE Trans. on Computers, Vol. C24, Sept. 1975,

861-867.

[Hs77] Hsieh, E.P., Rasmussen, R.A., Vidunas, L.J., Davis, W.T., "Delay Test Generation," Proc. 14th Design Automation Conference, New Orleans, Louisiana, June 1977, 486-491.

[Hu75] Hurtado, M., "Dynamic Structure and Performance of Asymptotically Bistable Systems," Washington University D.Sc. dissertation, 1975.

[Ki66] Kirkpatrick, T.I., and Clark, N.R., "PERT as an AID to Logic Design," IBM Journal of Research and Development, Vol. 10, March 1966, 135-141.

[Ko78] Koppel, A., Shah, S., and Puri, P., "A High Performance Delay Calculation Software System For MOSFET Digital Logic Chips," Proc. 15th Design Automation Conference, Las Vegas, Nev., June 1978, 405-417.

[Kr67] Krieger, M., Basic Switching Circuit Theory, The MacMillan Company, 1967.

[Kr77] Krohn, H.E., "Design Verification of Large Scientific Computers," Proc. 14th Design Automation Conference, June 1977, New Orleans, 377-385.

[Ku76] Kusik, R., and Wesley, P., "Hierarchical Logic Simulation for Digital Systems Development," Proc. Electro/76, Boston, Mass., May 1976, pp. 26.3.1-26.3.8.

[Li66] Littlefield, W.M., and Chaney, T.J., "The Glitch Phenomenon," Technical Memorandum #10, Washington University Computer Systems Laboratory, St. Louis, Mo., December 1966.

[Lo75] Losleben, P., "Design Validation in Hierarchical Systems," Proc. 12th Design Automation Conference, Boston, Mass., June 1975, 431-438.

[Ma77a] Magnhagen, B., "Practical Experiences From Signal Probability Simulation of Digital Designs," Proc. 14th Design Automation Conference, New Orleans, Louisiana, June 1977, 216-219.

[Ma77b] Magnhagen, B., "Probability Based Verification of Time Margins in Digital Designs," No. 17 in Linkoping Studies in Science and Technology. Dissertations., Department of Electrical Engineering, Linkoping University, Linkoping, Sweden, September 1977.

[Mc62] McCluskey, E.J. Jr., and Bartee, T.C., A Survey of Switching Circuit Theory, McGraw-Hill Book Company, 1962.

[Mc78a] McWilliams, T.M. and Widdoes, L.C., "SCALD: Structured Computer-Aided Logic Design," Proc. of the 15th Design Automation Conference, Las Vegas, Nev., June 1978, 271-277.

[Mc78b] McWilliams, T.M. and Widdoes, L.C., "The SCALD Physical Design Subsystem," Proc. 15th Design Automation Conference, Las Vegas, Nev., June 1978, 278-284.

[Me80] Mead, C. and Conway, L., Introduction to VLSI Systems, Addison-Wesley Publishing Company, 1980.

[No77] Noon, W.A., "A Design Verification and Logic Validation System," Proc. 14th Design Automation Conference, New Orleans, Louisiana, June 1977, 362-368.

[Ob70] Oberman, R.M.M., Disciplines in Combinational and Sequential Circuit Design, McGraw-Hill Book Company, 1970.

[Pa75] Parker, K.P., and McCluskey, E.J., "Probabilistic Treatment of General Combinational Networks," IEEE Trans. on Computers, Vol. C24, June 1975, 668-670.

[Pi72] Pilling, D.J., Ordung, T.F., and Heald, D., "Timing Delays in LSI Circuits," IEEE International Symposium on Circuit Theory, North Hollywood, Ca., April 1972, 311-315.

[Pi73] Pilling, D.J., Sun, H.B., "Computer-Aided Prediction of Delays in LSI Logic Systems," Proc. 10th Design Automation Conference, June 1973, 182-186.

[Ru73] Ruehli, A.E., "Electrical Considerations in the Computer-Aided Design of Logic Circuit Interconnections," Proc. 10th Design Automation Workship, June 1973, 262-266.

[Se79] Seitz, C.L., "Self-Timed VLSI Systems," Proc. of the Caltech Conference on VLSI, Jan. 1979.

[Sc69] Schnurmann, H.D., and Maling, K., "A Statistical Approach to the Computation of Delays in Logic Circuits," IEEE Trans. on Computers, Vol.

C-18, April 1969, 320-328.

[Sp78] S-1 Project Staff, "Advanced Digital Computing Technology Base Development for Navy Applications: The S-1 Project," Prepared for the Naval Systems Division, Office of Naval Research, September 30, 1978. (UCID-18038)

[Sp79] S-1 Project Staff, "FY79 Annual Report: The S-1 Project," Prepared for The Naval System Division, Office of Naval Research; The Command and Control Division, Naval Electronics Systems Command; and The Command, Control, Communication, and Intelligence Program Office, Naval Material Command. Work in part performed under the auspices of the U.S. Department of Energy under Contract No. W-7405-ENG-48, September 30, 1979. (UCID-18619)

[St77] Storey, T.M., and Barry, J.W., "Delay Test Simulation," Proc. 14th Design Automation Conference, New Orleans, Louisiana, June 1977, 492-494.

[Sz72] Szygenda, S.A., "TEGAS2--Anatomy of a General Purpose Test Generation and Simulation System for Digital Logic," Proc. 9th Design Automation Conference, June, 1972, 116-127.

[Sz75] Szygenda, S.A., and Thompson, E.W., "Digital Logic Simulation In a Time-Based, Table-Driven Environment; Part 1. Design Verification," IEEE Computer, Vol. 8, No. 3, March 1975, 24-36.

[Th74] Thompson, E.W., Szygenda, S.A., Billawala, N., Pierce, R., "Timing Analysis For Digital Fault Simulation Using Assignable Delays," Proc. 11th Design Automation Workshop, Denver, Colorado, June, 1974, 266-272.

[Un69] Unger, S.H., Asynchronous Sequential Switching Circuits, Wiley-Interscience, 1969.

[va77] vanCleemput, W.M., "An Hierarchical Language for the Structural Description of Digital Systems," Proc. 14th Design Automation Conference, June 1977, New Orleans, 377-385.

[Wo78] Wold, M.A., "Design Verification and Performance Analysis," Proc. 15th Design Automation Conference, Las Vegas, Nev., June 1978, 264-270.

GPO 689-102/5506